FEATool

Version: 1.8

User's Guide

Precise Simulation Ltd.
2003, 20F, Tower 5
China Hong Kong City
33 Canton Road
Tsim Sha Tsui, Kowloon
Hong Kong

Home page and email
www.featool.com
info@featool.com

(c) Copyright 2013-2018 by Precise Simulation Ltd. All rights reserved

The software described in this document is furnished under a license agreement. The software may be used, modified, and copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from Precise Simulation Ltd.

FEATool^{TM} and FEATool^{TM} Multiphysics are trademarks of Precise Simulation Limited.

MATLAB is a registered trademark of The MathWorks, Inc., and OpenFOAM is a registered trademark of ESI Group, other product or brand names are registered trademarks of their respective holders.

Contents

Contents iii

1 Graphical User Interface 2
- 1.1 Main GUI Window . . . 2
  - 1.1.1 Main GUI Axes . . . 3
  - 1.1.2 Menus . . . 3
  - 1.1.3 GUI Toolbar . . . 7
  - 1.1.4 Mode Buttons . . . 7
  - 1.1.5 Mode Tool Buttons . . . 8
  - 1.1.6 Output and Message Window . . . 9
  - 1.1.7 Command Input Window . . . 10
- 1.2 New Problem Dialog Box . . . 10
- 1.3 Other GUI Operations . . . 11
- 1.4 Axes/Grid Settings Dialog Box . . . 11
- 1.5 Keyboard Shortcuts . . . 12

2 Geometry 13
- 2.1 Geometry Mode . . . 13
- 2.2 Geometry Objects . . . 14
- 2.3 CAD Import and Export . . . 16
- 2.4 Geometry Operations . . . 16
- 2.5 Examples . . . 19
- 2.6 Command Reference . . . 19

3 Grid **21**
3.1 Grid Mode . . . 21
3.2 External Grid Generators . . . 24
3.2.1 Gmsh and Triangle Installation . . . 24
3.2.2 Usage Notes . . . 24
3.3 Grid Reference Material . . . 25
3.3.1 Grid Format . . . 25
3.3.2 Grid Cell Definitions . . . 26
3.3.3 Grid Example 1 . . . 30
3.3.4 Grid Example 2 . . . 31
3.3.5 Grid Example 3 . . . 33
3.3.6 Grid Import and Export . . . 34
3.3.7 Creating Structured Grids . . . 38
3.3.8 Quadrilateral Grid Generation . . . 42
3.3.9 Grid Utility Functions . . . 43

4 Physics **46**
4.1 Equation and Boundary Modes . . . 46
4.2 Multiphysics . . . 47
4.3 Physics Modes . . . 49
4.3.1 Poisson Equation . . . 50
4.3.2 Convection and Diffusion . . . 52
4.3.3 Conductive Media DC . . . 53
4.3.4 Electrostatics . . . 54
4.3.5 Magnetostatics . . . 55
4.3.6 Heat Transfer (Convection and Conduction) . . . 57
4.3.7 Euler-Bernoulli Beam . . . 59
4.3.8 Plane Stress . . . 60
4.3.9 Plane Strain . . . 62
4.3.10 Axisymmetric Stress-Strain . . . 64
4.3.11 Linear Elasticity . . . 65
4.3.12 Darcy's Law . . . 67
4.3.13 Brinkman Equations . . . 69
4.3.14 Navier-Stokes Equations (Incompressible Fluid Flow) . . . 70
4.3.15 Custom Equation . . . 72
4.4 Artificial Stabilization . . . 74

5 Solvers **75**
5.1 Solve Mode . . . 75
5.2 Solver Settings Dialog Box . . . 76
5.3 OpenFOAM CFD Solver . . . 78
5.3.1 OpenFOAM Usage . . . 78
5.3.2 Command Line (CLI) Usage . . . 79
5.3.3 Installation . . . 80
5.3.4 Limitations . . . 81
5.3.5 Advanced Usage . . . 81
5.3.6 Tutorial . . . 82
5.4 FEniCS General FEA Solver . . . 82
5.4.1 GUI Usage . . . 83
5.4.2 Command Line Usage . . . 84
5.4.3 Notes . . . 86
5.4.4 Installation . . . 87

6 Postprocessing **88**
6.1 Postprocessing Settings . . . 89
6.1.1 Surface Plot . . . 90
6.1.2 Height Expression . . . 90
6.1.3 Contour Plot . . . 91
6.1.4 Iso Plot . . . 92
6.1.5 Slice Plot . . . 92
6.1.6 Arrow Plot . . . 93
6.1.7 General Settings . . . 94
6.2 Advanced Postprocessing . . . 94
6.2.1 Min/Max Evaluation . . . 94
6.2.2 Point/Line Evaluation . . . 95
6.2.3 Boundary/Subdomain Integration . . . 96
6.2.4 Postprocessing Call . . . 97
6.2.5 Export Results . . . 97

7 Tutorials **98**
7.1 Heat Transfer - Shrink Fitting of an Assembly . . . 98
7.1.1 Heat Transfer - Shrink Fitting of an Assembly using the GUI . . . 100
7.1.2 Heat Transfer - Shrink Fitting of an Assembly using the CLI . . . 110
7.1.3 References . . . 110

7.2 Fluid Dynamics - Flow Around a Cylinder . . . 110
7.2.1 Flow Around a Cylinder using the GUI . . . 111
7.2.2 Flow Around a Cylinder using the CLI . . . 129
7.2.3 Instationary Flow Around a Cylinder . . . 129
7.2.4 References . . . 129
7.3 Fluid Dynamics - Axisymmetric Fluid Flow . . . 131
7.4 Fluid Dynamics - Flow Over a Backwards Facing Step . . . 131
7.4.1 Flow over a Backwards Facing Step with the external OpenFOAM CFD Solver . . . 131
7.4.2 Flow over a Backwards Facing Step using the CLI . . . 143
7.4.3 References . . . 143
7.5 Structural Mechanics - Thin Plate with Hole . . . 143
7.6 Structural Mechanics - 3D Deflection of a Bracket . . . 144
7.6.1 3D Deflection of a Bracket using the GUI . . . 144
7.6.2 3D Deflection of a Bracket using the CLI . . . 160
7.6.3 Parametric Study of Deflection of a Bracket . . . 161
7.7 Electromagnetics - Electrostatic Spherical Capacitor . . . 166
7.7.1 Electrostatic Spherical Capacitor using the CLI . . . 169
7.8 Electromagnetics - Magnetic Field Around a Horseshoe Magnet . . . 170
7.8.1 Magnetic Field Around a Horseshoe Magnet using the CLI . . . 173
7.9 Multiphysics - Natural Convection in a Square Cavity . . . 174
7.9.1 Natural Convection in a Square Cavity using the GUI . . . 174
7.9.2 Natural Convection in a Square Cavity using the CLI . . . 191
7.9.3 References . . . 191
7.10 Multiphysics - Heat Exchanger . . . 191
7.11 Multiphysics - Piezoelectric Bending of a Beam . . . 192
7.11.1 References . . . 196
7.12 Custom Equation - Black-Scholes model equation . . . 197
7.12.1 Black-Scholes model equation using the GUI . . . 197
7.12.2 Black-Scholes model equation using the CLI . . . 207
7.13 Custom Equation - Wave Equation . . . 207
7.14 Classic Equation - Poisson Equation on a Circle . . . 208
7.14.1 Poisson Equation using the GUI . . . 208
7.14.2 Poisson Equation using the CLI and Physics Modes . . . 221
7.14.3 Poisson Equation using the CLI without Physics Modes . . . 224
7.14.4 Poisson Equation using the CLI and core FEM library functions . . . 227

7.15 Classic Equation - Poisson Equation with a Point Source . . . 231
7.16 Classic Equation - Poisson Equation for a Sphere . . . 231
7.17 Other Examples . . . 235

8 Mathematical Modeling 240
8.1 Conservation Laws . . . 240
8.2 Poisson Equation . . . 243
8.3 Boundary and Initial Conditions . . . 244
8.4 Poisson Example . . . 244

9 Problem Definition 245
9.1 Problem Definition Struct . . . 245
9.1.1 Equation Struct Specification . . . 248
9.1.2 Boundary Struct Specification . . . 249
9.1.3 Point Struct Specification . . . 251

10 Function Index 253
10.1 File List . . . 253

11 Function Reference 275
11.1 addpaths . . . 275
11.2 core/asmgroups . . . 275
11.3 core/asmrowcolptr . . . 276
11.4 core/assemblea . . . 276
11.5 core/assemblef . . . 278
11.6 core/assemblej . . . 279
11.7 core/assemblenjac . . . 281
11.8 core/assembleprob . . . 281
11.9 core/assemjac . . . 282
11.10 core/assemmat . . . 283
11.11 core/bdrneu . . . 284
11.12 core/bdrneu_lin . . . 284
11.13 core/bdrsetd . . . 285
11.14 core/bdrcpl . . . 285
11.15 core/bdrslip . . . 286
11.16 core/cellvol . . . 286
11.17 core/coordr2l . . . 286

11.18 core/coordtf . . . 287
11.19 core/quadrule . . . 287
11.20 core/quadruleb . . . 288
11.21 core/quadefc . . . 288
11.22 core/evaldvar . . . 289
11.23 core/evalexpr0 . . . 289
11.24 core/evalexprc . . . 290
11.25 core/evalexprp . . . 290
11.26 core/evalinit . . . 291
11.27 core/evalsfun . . . 291
11.28 core/lcoorde . . . 292
11.29 core/mapdofbdr . . . 292
11.30 core/re2locquad . . . 293
11.31 core/solve_hook . . . 293
11.32 core/pntset . . . 294
11.33 core/pntsetf . . . 294
11.34 core/subcoef . . . 295
11.35 core/tfjac . . . 295
11.36 core/tfjachex . . . 296
11.37 core/tfjacline . . . 297
11.38 core/tfjacquad . . . 297
11.39 core/tfjactet . . . 298
11.40 core/tfjactri . . . 299
11.41 ellib/sf_disc0 . . . 299
11.42 ellib/sf_disc1 . . . 301
11.43 ellib/sf_hex_Q1 . . . 302
11.44 ellib/sf_hex_Q1nc . . . 307
11.45 ellib/sf_hex_Q2 . . . 312
11.46 ellib/sf_hex_Q3 . . . 323
11.47 ellib/sf_hex_Q4 . . . 332
11.48 ellib/sf_hex_Q5 . . . 349
11.49 ellib/sf_line_H3 . . . 376
11.50 ellib/sf_line_P3 . . . 378
11.51 ellib/sf_line_P4 . . . 380
11.52 ellib/sf_line_P5 . . . 383
11.53 ellib/sf_quad_H3 . . . 385

11.54 ellib/sf_quad_Q1 . . . 389
11.55 ellib/sf_quad_Q1nc . . . 392
11.56 ellib/sf_quad_Q2 . . . 395
11.57 ellib/sf_quad_Q3 . . . 398
11.58 ellib/sf_quad_Q4 . . . 402
11.59 ellib/sf_quad_Q5 . . . 406
11.60 ellib/sf_simp_P1 . . . 411
11.61 ellib/sf_simp_P1nc . . . 412
11.62 ellib/sf_simp_P2 . . . 413
11.63 ellib/sf_simp_P2bub . . . 416
11.64 ellib/sf_tet_P3 . . . 420
11.65 ellib/sf_tet_P4 . . . 432
11.66 ellib/sf_tet_P5 . . . 452
11.67 ellib/sf_tri_H3 . . . 481
11.68 ellib/sf_tri_P3 . . . 484
11.69 ellib/sf_tri_P4 . . . 489
11.70 ellib/sf_tri_P5 . . . 496
11.71 ellib/sflag1 . . . 505
11.72 ellib/sflag2 . . . 506
11.73 ellib/sflag3 . . . 507
11.74 ellib/sflag4 . . . 509
11.75 ellib/sflag5 . . . 511
11.76 ellib/sfnoncon1 . . . 512
11.77 ellib/sfnonconp1 . . . 513
11.78 ellib/lcoord_cell . . . 515
11.79 ellib/grid_cell_type . . . 519
11.80 evalexpr . . . 520
11.81 examples/ex_brinkman1 . . . 521
11.82 examples/ex_convdiff1 . . . 523
11.83 examples/ex_convdiff2 . . . 526
11.84 examples/ex_convdiff3 . . . 529
11.85 examples/ex_convdiff4 . . . 531
11.86 examples/ex_convdiff5 . . . 534
11.87 examples/ex_convdiff6 . . . 536
11.88 examples/ex_convreact1 . . . 539
11.89 examples/ex_custom_equation1 . . . 542

11.90 examples/ex_euler_beam1 . . . 544
11.91 examples/ex_euler_beam2 . . . 547
11.92 examples/ex_nonlinear_pde1 . . . 549
11.93 examples/ex_nonlinear_pde2 . . . 551
11.94 examples/ex_diffusion1 . . . 554
11.95 examples/ex_diffusion2 . . . 557
11.96 examples/ex_heattransfer1 . . . 560
11.97 examples/ex_heattransfer2 . . . 563
11.98 examples/ex_heattransfer3 . . . 565
11.99 examples/ex_heattransfer4 . . . 568
11.100 examples/ex_heattransfer5 . . . 570
11.101 examples/ex_heattransfer6 . . . 573
11.102 examples/ex_heattransfer7 . . . 576
11.103 examples/ex_laplace1 . . . 578
11.104 examples/ex_laplace2 . . . 581
11.105 examples/ex_linearelasticity1 . . . 583
11.106 examples/ex_linearelasticity2 . . . 586
11.107 examples/ex_linearelasticity3 . . . 589
11.108 examples/ex_linearelasticity4 . . . 591
11.109 examples/ex_spanner . . . 594
11.110 examples/ex_multiphase1 . . . 596
11.111 examples/ex_multiphase2 . . . 599
11.112 examples/ex_multiphase3 . . . 602
11.113 examples/ex_heat_exchanger1 . . . 605
11.114 examples/ex_natural_convection . . . 608
11.115 examples/ex_darcy1 . . . 612
11.116 examples/ex_navierstokes1 . . . 614
11.117 examples/ex_navierstokes2 . . . 618
11.118 examples/ex_navierstokes3 . . . 622
11.119 examples/ex_navierstokes4 . . . 626
11.120 examples/ex_navierstokes5 . . . 630
11.121 examples/ex_navierstokes6 . . . 633
11.122 examples/ex_navierstokes7 . . . 636
11.123 examples/ex_navierstokes8 . . . 638
11.124 examples/ex_navierstokes9 . . . 640
11.125 examples/ex_navierstokes10 . . . 643

11.126 examples/ex_navierstokes11 . . . 645
11.127 examples/ex_navierstokes12 . . . 647
11.128 examples/ex_navierstokes13 . . . 649
11.129 examples/ex_navierstokes14 . . . 652
11.130 examples/ex_swirl_flow1 . . . 655
11.131 examples/ex_swirl_flow2 . . . 658
11.132 examples/ex_swirl_flow3 . . . 660
11.133 examples/ex_swirl_flow4 . . . 663
11.134 examples/ex_piezoelectric1 . . . 666
11.135 examples/ex_electrostatics1 . . . 670
11.136 examples/ex_electrostatics2 . . . 672
11.137 examples/ex_resistive_heating1 . . . 674
11.138 examples/ex_magnetostatics1 . . . 677
11.139 examples/ex_magnetostatics2 . . . 679
11.140 examples/ex_magnetostatics3 . . . 682
11.141 examples/ex_magnetostatics4 . . . 683
11.142 examples/ex_magnetostatics5 . . . 685
11.143 examples/ex_planestress1 . . . 687
11.144 examples/ex_planestress2 . . . 690
11.145 examples/ex_planestress3 . . . 692
11.146 examples/ex_planestress4 . . . 695
11.147 examples/ex_planestress5 . . . 698
11.148 examples/ex_planestress6 . . . 700
11.149 examples/ex_planestrain1 . . . 703
11.150 examples/ex_axistressstrain1 . . . 706
11.151 examples/ex_axistressstrain2 . . . 708
11.152 examples/ex_axistressstrain3 . . . 710
11.153 examples/ex_axistressstrain4 . . . 713
11.154 examples/ex_poisson1 . . . 718
11.155 examples/ex_poisson2 . . . 722
11.156 examples/ex_poisson3 . . . 725
11.157 examples/ex_poisson3_derivative . . . 728
11.158 examples/ex_poisson4 . . . 728
11.159 examples/ex_poisson5 . . . 731
11.160 examples/ex_poisson6 . . . 734
11.161 examples/ex_poisson7 . . . 737

11.162 examples/ex_periodic1 . . . 739
11.163 examples/ex_periodic2 . . . 741
11.164 examples/ex_robinbc1 . . . 744
11.165 examples/ex_waveequation1 . . . 745
11.166 featool . . . 749
11.167 fenics/fenics . . . 749
11.168 fenics/fenics_gui . . . 750
11.169 openfoam/openfoam . . . 750
11.170 openfoam/openfoam_data . . . 750
11.171 openfoam/openfoam_gui . . . 751
11.172 openfoam/openfoam_write . . . 751
11.173 geom/compute_distance . . . 752
11.174 geom/extract_entries . . . 752
11.175 geom/geom_add_gobj . . . 752
11.176 geom/geom_boundary_edges . . . 752
11.177 geom/geom_combine_bbox . . . 753
11.178 geom/geom_decompose_boundaries2 . . . 753
11.179 geom/geom_decompose_boundaries3 . . . 753
11.180 geom/geom_get_tags . . . 753
11.181 geom/geom_intersect_lines . . . 753
11.182 geom/geom_parse_formula . . . 754
11.183 geom/geom_restrict_bbox . . . 754
11.184 geom/geom_split_lines . . . 754
11.185 geom/geom_split_object . . . 755
11.186 geom/geom_uniqify_tag . . . 755
11.187 geom/geom_update_tags . . . 755
11.188 geom/gobj_boundary_vertbbox . . . 755
11.189 geom/gobj_reorient_boundaries . . . 755
11.190 geom/order_boundaries . . . 756
11.191 geom/plotgeom1 . . . 756
11.192 geom/plotgeom2 . . . 756
11.193 geom/plotgeom3 . . . 757
11.194 geom/plotgeomb . . . 758
11.195 geom/plotpoint . . . 758
11.196 geom/polyarea3 . . . 759
11.197 geom/reconstruct_boundaries . . . 759

11.198 geom/reconstruct_triangles . . . 759
11.199 geom2/bbox_intersect . . . 759
11.200 geom2/boundary_retriangulation . . . 759
11.201 geom2/compress_boundary_edges . . . 760
11.202 geom2/convert_gobj_polygons . . . 760
11.203 geom2/csg_op . . . 760
11.204 geom2/csg_polygon_recombination . . . 761
11.205 geom2/csg_polygon_tesselation . . . 762
11.206 geom2/geom_analyze . . . 762
11.207 geom2/geom_apply_formula . . . 762
11.208 geom2/geom_apply_transformation . . . 762
11.209 geom2/geom_combine_objects . . . 763
11.210 geom2/geom_decompose . . . 763
11.211 geom2/gobj_apply_op . . . 764
11.212 geom2/gobj_block . . . 764
11.213 geom2/gobj_circle . . . 765
11.214 geom2/gobj_cylinder . . . 765
11.215 geom2/gobj_decompose . . . 766
11.216 geom2/gobj_ellipse . . . 766
11.217 geom2/gobj_empty . . . 767
11.218 geom2/gobj_grid . . . 767
11.219 geom2/gobj_line . . . 767
11.220 geom2/gobj_point . . . 768
11.221 geom2/gobj_polygon . . . 768
11.222 geom2/gobj_rectangle . . . 768
11.223 geom2/gobj_sphere . . . 769
11.224 geom2/match_rows . . . 769
11.225 grid/assign_bdr . . . 769
11.226 grid/blockgrid . . . 770
11.227 grid/constrain_edges . . . 770
11.228 grid/circgrid . . . 771
11.229 grid/cylgrid . . . 772
11.230 grid/delcells . . . 772
11.231 grid/findbdr . . . 773
11.232 grid/gridadj . . . 773
11.233 grid/gridalign2 . . . 774

11.234 grid/gridbdr . . . 774
11.235 grid/gridbdr1 . . . 775
11.236 grid/gridbdrx . . . 776
11.237 grid/gridbfstf . . . 776
11.238 grid/gridcheck . . . 777
11.239 grid/gridconne . . . 777
11.240 grid/gridedge . . . 777
11.241 grid/gridextrude . . . 778
11.242 grid/gridface . . . 779
11.243 grid/gridmerge . . . 780
11.244 grid/gridnormals . . . 781
11.245 grid/gridrefine . . . 781
11.246 grid/gridrefine1 . . . 782
11.247 grid/gridrefine2 . . . 782
11.248 grid/gridrefine3 . . . 783
11.249 grid/gridrevolve . . . 783
11.250 grid/gridrotate . . . 784
11.251 grid/gridscale . . . 785
11.252 grid/gridsmooth . . . 785
11.253 grid/gridvert . . . 786
11.254 grid/grid_boundary_nodes . . . 787
11.255 impexp/geom2geo . . . 787
11.256 impexp/geom2poly . . . 787
11.257 impexp/grid2foam . . . 788
11.258 impexp/impexp_ascii . . . 788
11.259 impexp/impexp_dolfin . . . 789
11.260 impexp/impexp_foam . . . 789
11.261 impexp/impexp_gid . . . 790
11.262 impexp/impexp_gmsh . . . 790
11.263 impexp/impexp_gmv . . . 791
11.264 impexp/impexp_stl . . . 791
11.265 impexp/impexp_triangle . . . 792
11.266 impexp/log_message . . . 792
11.267 impexp/open_file . . . 792
11.268 grid/hex2tet . . . 792
11.269 grid/holegrid . . . 793

11.270 grid/linegrid . . . 794
11.271 grid/mesh_boundaries . . . 795
11.272 grid/mesh_size . . . 795
11.273 grid/rectgrid . . . 795
11.274 grid/reorient_cells . . . 796
11.275 grid/ringgrid . . . 796
11.276 grid/project_bdrp . . . 797
11.277 grid/quad2tri . . . 797
11.278 grid/quadmesh . . . 798
11.279 grid/reconstruct_bdr . . . 799
11.280 grid/selcells . . . 799
11.281 grid/spheregrid . . . 800
11.282 grid/searchgraph . . . 801
11.283 grid/tet2hex . . . 801
11.284 grid/tri2quad . . . 801
11.285 gridgen . . . 802
11.286 grid/gridgen_distmesh . . . 804
11.287 grid/gridgen_gmsh . . . 805
11.288 grid/gridgen_quad . . . 808
11.289 grid/gridgen_triangle . . . 809
11.290 grid/gridstat . . . 812
11.291 install . . . 813
11.292 intsubd . . . 813
11.293 intbdr . . . 813
11.294 minmaxsubd . . . 814
11.295 minmaxbdr . . . 815
11.296 physmodes/addphys . . . 815
11.297 physmodes/axistressstrain . . . 816
11.298 physmodes/brinkmaneqns . . . 816
11.299 physmodes/conductivemediadc . . . 816
11.300 physmodes/eulerbeam . . . 816
11.301 physmodes/electrostatics . . . 816
11.302 physmodes/magnetostatics . . . 816
11.303 physmodes/convectiondiffusion . . . 817
11.304 physmodes/customeqn . . . 817
11.305 physmodes/darcyslaw . . . 817

11.306 physmodes/heattransfer . . . 817
11.307 physmodes/linearelasticity . . . 817
11.308 physmodes/list_physmodes . . . 817
11.309 physmodes/navierstokes . . . 817
11.310 physmodes/physmode . . . 817
11.311 physmodes/planestrain . . . 818
11.312 physmodes/planestress . . . 818
11.313 physmodes/poisson . . . 818
11.314 plotbdr . . . 818
11.315 plotgeom . . . 819
11.316 plotgrid . . . 819
11.317 plotsubd . . . 820
11.318 post/contour2 . . . 821
11.319 post/postplot1 . . . 821
11.320 post/postplot2 . . . 822
11.321 post/postplot3 . . . 822
11.322 post/princse . . . 823
11.323 post/setaxes . . . 823
11.324 post/slice3 . . . 824
11.325 postplot . . . 824
11.326 solvestat . . . 825
11.327 solvetime . . . 826
11.328 uninstall . . . 827
11.329 util/conngrp . . . 827
11.330 util/cstrcmp . . . 827
11.331 util/deduplicate . . . 828
11.332 util/dehtmlify . . . 828
11.333 util/expandeqn . . . 828
11.334 util/findparblock . . . 829
11.335 util/interpl . . . 829
11.336 util/mergeeqncoefs . . . 829
11.337 util/parsecoef . . . 829
11.338 util/parseeqn . . . 830
11.339 util/parseexpr . . . 831
11.340 util/parseopt . . . 832
11.341 util/parsephys . . . 832

11.342 util/parseprob . . . 832
11.343 util/parseprob0 . . . 833
11.344 util/recursive_termsplit . . . 833
11.345 util/remove_duplicate_rows . . . 834
11.346 util/remove_parentheses . . . 834
11.347 util/rmkdir . . . 834
11.348 util/set_geomlib . . . 834
11.349 util/stfldcheck . . . 835
11.350 util/strconv . . . 835
11.351 util/stringify . . . 835
11.352 util/symvarbe . . . 836
11.353 util/termsplit . . . 836
11.354 util/timerfun . . . 836
11.355 util/urlget . . . 837
11.356 util/uunique . . . 837

FEATool

Welcome to FEATool^{TM} (short for Finite Element Analysis Toolbox), a GNU Octave and Matlab® toolbox for modeling and simulation of physics, partial differential equations (PDE), and mathematical problems with the finite element method (FEM).

FEATool aims to provide an easy to use and comprehensive all-in-one integrated simulation package for all types of finite element and multiphysics analyses, and combine the best of ease of use, powerful functionality, and extensibility for both beginners and advanced users. Features such as an intuitive and easy to learn graphical user interface (GUI), a complete library of grid generation, FEM, and postprocessing functions, as well as command line interface (CLI) programming, and interactive and interpreted programming and scripting capabilities, makes FEATool suitable for everyone from students learning mathematical modeling, to professionals and engineers wishing to explore new ideas in a simple, fast, and convenient way.

Please enjoy working with FEATool, and do not hesitate to get in contact if you have any questions or suggestions for improvements.

1 Graphical User Interface

This section describes how to use the FEATool GUI (available with Matlab or Octave version 4.0 or later). Please note that the GUI may look different depending on your system configuration and the Octave or Matlab version running FEATool.

1.1 Main GUI Window

After launching FEATool in Octave or Matlab (by typing *featool* at the command prompt) the GUI will load. The main GUI window which will open is composed of the main GUI axes, top menu and toolbar, left vertical mode and tool button bars, and also input, output, and terminal command windows. The function of these is explained below.

1.1.1 Main GUI Axes

The main GUI axes is where the geometry, computational grid, and postprocessing results are displayed. The view can be controlled by the **Zoom In** , **Zoom Out** , **Pan** , and **Rotate** buttons in the top toolbar.

1.1.2 Menus

On the top of the GUI toolbar there are menus for file, options, mode, and help operations. The function of these menus are described in detail below

The **File** menu features the following options:

- **New...** clears the old problem data and opens the *New Problem* dialog box.
- **Open...** allows for opening and loading a previously saved model (*.fea* file).
- **Save As...** allows for saving the current state of a model (as a *.fea* file).
- **Save As M-Script Model...** allows for saving the commands used in the session as a m-script model file (*.m* file). Note that m-script models does not save the grids or solutions, just the FEATool function calls that created them.
- **Clear M-Script Log** Clears the m-script command log.
- **Import** > **Variables From Main Workspace...** opens a dialog box which allows variables to be selected and imported from the Octave or Matlab workspace.
- **Export** > **Variables To Main Workspace...** opens a dialog box which allows variables to be selected and exported to the main workspace.
- **Export** > **FEA Problem Struct To Main Workspace...** exports the FEATool problem struct to the main workspace with the variable name *fea*.
- **Exit** closes and exits the FEATool GUI.

The **Options** menu firstly allows the axes and grid settings to be modified (these settings are described in the axes/grid settings section below). The FEATool GUI options can be inspected

and edited by calling the *dlg_Options* function which opens the Options dialog box (this will effectively edit the *guiparam.txt* parameter file located in the *featool/gui* folder).

The **Geometry** menu allows for switching to **Geometry Mode**, adding specific points **Add Point...**, and also accessing the **Combine Objects...** option which allows specification of a complex formula for geometry object composition. Geometry export and import between STL CAD, Gmsh *geo*, and Triangle *poly* formats is also supported.

The **Grid** menu allows for switching to **Grid Mode** as well as accessing and specifying the **Grid Generation Command...**. **Convert Grid Cells** converts between triangular and quadrilateral cells in 2D, and between tetrahedral and hexahedral cells in 3D.

Grid Mode also supports import and export of grids through external ASCII format files, these option can also be accessed from the **Grid** menu

- **Import Grid** > **FEniCS/Dolfin Format...** imports a grid in `FEniCS/Dolfin` format (.xml file) (calls impexp_dolfin).
- **Import Grid** > **GiD Format...** imports a grid in `GiD` format (.gid file) (calls impexp_gid).
- **Import Grid** > **Gmsh Format...** imports a grid in `Gmsh` format (.msh file) (calls impexp_gmsh).
- **Import Grid** > **GMV Format...** imports a grid in `General Mesh Viewer` (GMV) format (.gmv file) (calls impexp_gmv).
- **Import Grid** > **Triangle Format...** imports a 2D `Triangle` grid (.node/.ele files) (calls impexp_triangle).

- **Export Grid** > **FEniCS/Dolfin Format...** exports a grid in `FEniCS/Dolfin` format (.xml file) (calls impexp_dolfin).
- **Export Grid** > **GiD Format...** exports a grid in `GiD` format (.gid file) (calls impexp_gid).
- **Export Grid** > **Gmsh Format...** exports a grid in `Gmsh` format (.msh file) (calls impexp_gmsh).

- **Export Grid** > **GMV Format...** exports a grid in `General Mesh Viewer` (GMV) format (.gmv file) (calls impexp_gmv).

- **Export Grid** > **OpenFOAM Format...** exports a grid in `OpenFOAM` format (calls grid2foam).

- **Export Grid** > **Triangle Format...** exports a 2D `Triangle` polygonal geometry description (.poly file) (calls impexp_triangle).

In grid mode the overall global maximum grid cell size variable *hmax* can be changed with the **Grid Size** edit field and slider in the grid tools toolbar.

The **Equation** and **Boundary** menus allows for switching to **Equation/Boundary Modes** and also opening the **Equation/Boundary Settings...** dialog boxes (see the equation and boundary mode sections). Moreover, the **Model Constant and Expressions...** option allows for input of modeling variables and expressions that can be used in equations and boundary settings, and postprocessing. Point sources and constraints can also be added here with the corresponding **Add Point Sources...** and **Add Point Constraints...** menu options.

The **Solve** menu similarly allows for switching to **Solve Mode** and also opening the **Solver Settings...** dialog box (these settings are described in the solver settings section). This menu also features an option to **Get Initial Solution** which computes the expressions defined in the equation settings *Initial Condition* fields and returns this as the solution. Moreover, the solver call command can be changed with the **Solver Call Command...** option.

The **Post** menu allows for switching to **Postprocessing Mode**, and opening the **Postprocessing Settings...** dialog box (see the postprocessing settings section below). In addition there are options for evaluating the minima and maxima for general expressions with **Min/Max Evaluation...**, points and line evaluation **Point/Line Evaluation...**, and integration of postprocessing expressions over subdomains **Subdomain Integration...** (calling intsubd) and boundaries **Boundary Integration...** (calling intbdr). The postprocessing call command can be inspected and changed with the **Postprocessing Command...** option. Moreover, the solution and postprocessing data can also be exported to be used with the `General Mesh Viewer` (GMV) through the **Export Results...** > **GMV Format...** menu option.

The **Help** menu opens the **About...** information dialog box and shows FEATool version information, and selecting **Register FEATool...** allows entering a FEATool license key. Moreover,

there is also a link to open the **FEATool Documentation** and visit the online FEATool Articles and Blog archive.

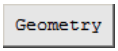
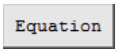
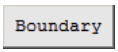
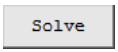
1.1.3 GUI Toolbar

The top GUI toolbar contains the following shortcut buttons (note that the GUI might show different options and icons depending on the Octave or Matlab version used)

- *New Problem* starts a new problem definition
- *Open Problem* loads a FEATool model
- *Save Problem* saves the current FEATool model
- *Zoom In* the main view axes
- *Zoom Out* the main view axes
- *Pan* around the main view axes
- *Rotate* the main view axes (only in 3D)

1.1.4 Mode Buttons

To the left of the main view axes is the mode toolbar which consists of six mode buttons which switches between the different modes

-  Switches to geometry mode
- Switches to grid generation mode
-  Switches to equation/subdomain specification mode (and opens the corresponding dialog box)
-  Switches to boundary settings mode (and opens the corresponding dialog box)
-  Switches to solve mode
- Switches to postprocessing mode

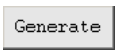
1.1.5 Mode Tool Buttons

Below these buttons are the mode tool buttons which can change depending on which mode is selected.

Geometry Mode

- Creates a square/rectangle
- Creates a circle/ellipse
- Creates a polygon
- Creates a block
- Creates a sphere
- Creates a cylinder
- Join selected geometry objects
- Subtract geometry objects from first selected
- Intersection of geometry objects
- Undo or split composite or transformed geometry object
- Rotate, scale, and move selected geometry object
- Delete selected geometry objects
- Inspect/edit properties of selected geometry object

Grid Mode

- Creates a line grid (only in 1D)
- 

Initialize an unstructured grid

- Refine Uniform grid refinement
- Settings Advanced grid settings and grid generation algorithm selection

Equation Mode

- Open the equation/subdomain settings dialog box
- Open the constants and expressions dialog box

Boundary Mode

- Open the boundary settings dialog box

Solve Mode

- Open solver settings dialog box
- Solve problem with initial conditions prescribed from equation settings
- Solve problem with last solution as initial value
- Solve fluid flow problem with the OpenFOAM external CFD solver
- Solve multiphysics problem FEniCS external solver

Post Mode

- Open the postprocessing settings dialog box
- Share Open dialog box for sharing results to Facebook, LinkedIn, Twitter and, `Plotly` web plot and jpeg/png image generation

1.1.6 Output and Message Window

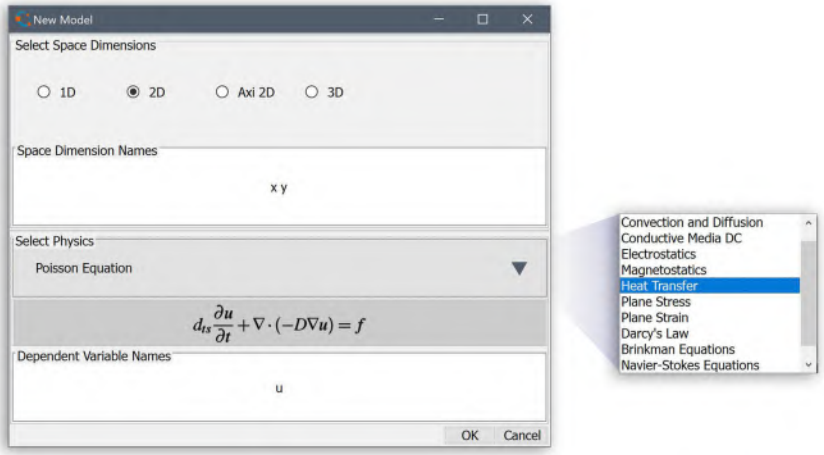
Below the main GUI axes is the output terminal window which displays messages, and output from the local FEATool workspace including output from the command input window below.

1.1.7 Command Input Window

On the bottom is the command input window which allows Matlab commands to be operated in the FEATool workspace (note that although the FEATool *fea* problem struct can be inspected and edited it might break and cause errors in the GUI operation).

1.2 New Problem Dialog Box

Pressing the *New Problem* button or selecting **New...** from the **File** menu starts a new problem definition and opens the *New Problem* dialog box.



The *New Problem* dialog box firstly allows for selecting the number of space dimensions to work in. The *Select Space Dimensions* frame allows one to choose from **1D**, **2D**, **Axi 2D** (for axisymmetric/cylindrical pseudo three-dimensional coordinate system), and **3D**. Depending on which selection is made the default *Space Dimension Names* (**x**, **y**, and **z**, or alternatively **r** and **z** in cylindrical coordinates) will be shown in the corresponding edit field. The space dimension or independent variable names can be changed if desired, and are used in FEATool defining equation, boundary, postprocessing, and other expressions.

The *Select Physics* drop down box shows a list of pre-defined physics application modes available for the selected space dimension. By choosing one of the physics modes the corresponding physics equation and default *Dependent Variable Names* will be updated. This will be the first physics mode defining a new model. The dependent variable names can also be changed as long as they do not conflict with each other or other physics modes. Additional physics modes can be added later by using the **+** tab in the **Equation Settings** dialog box in *Subdomain Mode*, as described the Multiphysics section.

To finish the initial selection process and start a new model press **OK**, or press **Cancel** to return to the previous model.

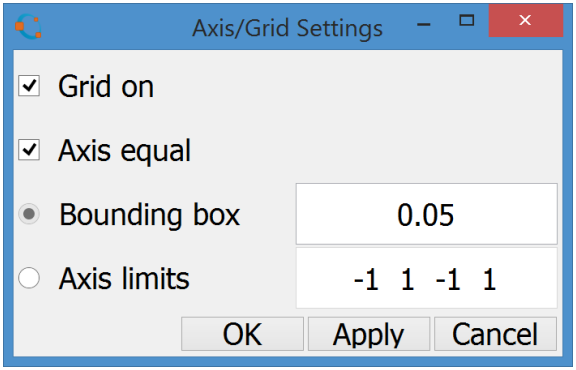
1.3 Other GUI Operations

Equation, Boundary, Solve, and Postprocessing mode usage and corresponding dialog boxes are described in the following sections.

- Physics Modes
- Solvers
- Postprocessing

1.4 Axes/Grid Settings Dialog Box

In the **Axis/Grid Settings** dialog box the **Grid on** option turns on and off the grid. By having **Axis equal** checked the x and-axes will have equal proportional lengths. **Bounding box** indicates how much extra relative space should be shown apart from the geometry. Alternatively, one can manually set the **Axis limits**. In 3D a slider control allows setting the transparency (opacity) of geometry objects.



1.5 Keyboard Shortcuts

The following keyboard shortcuts are available in FEATool:

- **Ctrl + n** Open the new problem dialog box to start a new model.
- **Ctrl + o** Open a previously saved FEA problem model.
- **Ctrl + s** Open Save FEA Model dialog box.
- **Ctrl + a** Select all geometry objects in geometry mode, also works in list boxes and edit fields.
- **Ctrl + d** Clear selected objects.

Moreover, in 2D Geometry mode one can hold down a **Shift** key while using the **Create circle/ellipse** tool to restrict to a circular geometry instead of allowing elliptical ones. In the FEATool **Command Window** prompt it is also possible to use the *up* and *down* arrow keys to scroll backwards and forwards through the command log history, and *Ctrl + c* to clear the command input edit field.

2 Geometry

The first step in the modeling process is to create a geometry to define the domain to be simulated. This section describes how a complex geometry can be created in FEATool by combining geometry objects such as rectangles, circles, polygons, blocks, and cylinders.

2.1 Geometry Mode

Geometry mode is the default mode that a new FEATool problem starts in (except for 1D problems where a geometry is unnecessary). This mode can also be selected by pressing the

mode button or corresponding menu option. While in geometry mode the toolbar buttons represent the geometry objects and operations that are available. The main gui window will also show all present geometry objects and highlight them in red while they are selected. A selection list box will also be present which is used to select geometry object to perform and apply operations on (in 2D geometry objects can also be selected by clicking on them in the main gui window).

2.2 Geometry Objects

In two dimensions rectangles, squares, ellipses, circles, and polygonal geometry object primitives are available. To create a 2D geometry object click on one of the corresponding toolbar buttons

-

  To create rectangles and squares, left mouse click on a point in the main gui axes to represent one corner, then hold the mouse button and move the mouse pointer to the other corresponding corner. The resulting shape will be highlighted in red lines. Release the mouse button to finalize the geometry object.

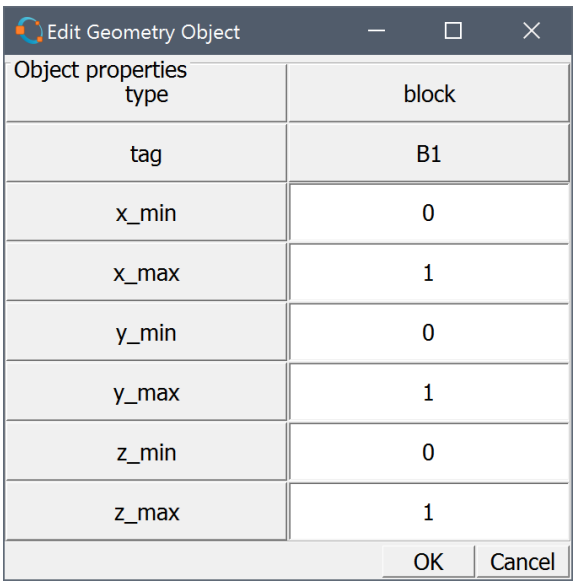
- For circles and ellipses, left mouse click on a point in the main gui axes to represent the centroid, then hold the mouse button and move the mouse pointer to a desired shape (highlighted in red lines). Release the mouse button to finalize the geometry object (holding the shift button will restrict the shape to a circle).

- To create a polygon add new points with left mouse clicks. The polygon lines will be highlighted in red. Finish with right mouse click.

In three dimensions (3D) block, cylinder, and sphere, geometry object primitives are available. To create a 3D geometry object click on one of the corresponding toolbar buttons

- Creates a block
- Creates a sphere
- Creates a cylinder

In three dimensions geometry objects are created by specifying the corresponding dimensions. Geometry object properties and dimensions can also be edited by using the *Inspect/edit selected geometry object* option. Pressing the button will open a dialog box for the selected geometry object where object properties can be edited.



The geometry menu also features a point object. By selecting the **Add Point...** menu option one can specify the point coordinates for the objects. Point objects are used to enforce grid vertices at specific points in the geometry and are only used in the grid generation process.

2.3 CAD Import and Export

Import and export of CAD geometries in the STL ASCII format is supported through **Geometry** menu options. Geometry export for Gmsh (geo) and Triangle (poly) ASCII formats is also supported.

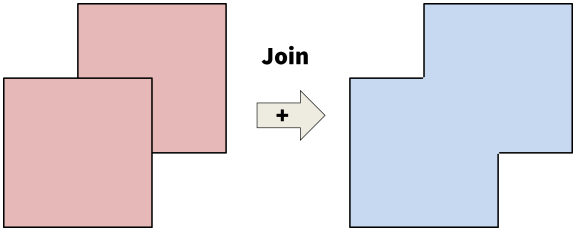
STL CAD file import is supported for both full 3D and (pseudo) 2D geometries. If all coordinates as in the same plane the geometry will be considered two-dimensional and treated correspondingly (typically takes as z-coordinates equal to zero). Note that 3D geometries can only be imported into models started with a 3D coordinate system, and vice versa.

The STL import function will try to coalesce duplicated points and join triangles to a solid geometry object. If this process fails, consider CAD format repair in a dedicated external software prior to import. Moreover, FEATool will TRY to automatically split the object into distinct boundaries. To manually specify boundary patches it is possible to import several STL files/segments where each STL group will be treated separately. Manual command line interface CLI usage is also possible directly with the impexp_stl function allowing for tuning of import parameters. An example utilizing CAD geometry import functionality can be found as ex_spanner in the examples directory.

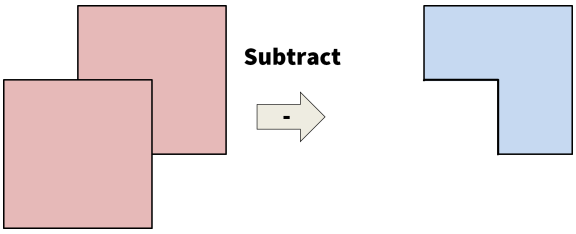
2.4 Geometry Operations

The following operations allow geometry objects to be combined (join and intersection operations) or subtracted to form more complex geometries.

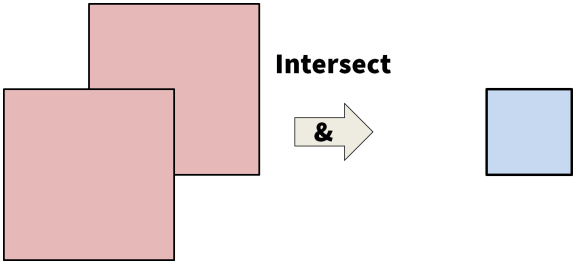
- The join operation merges all selected geometry objects to a single composite object.



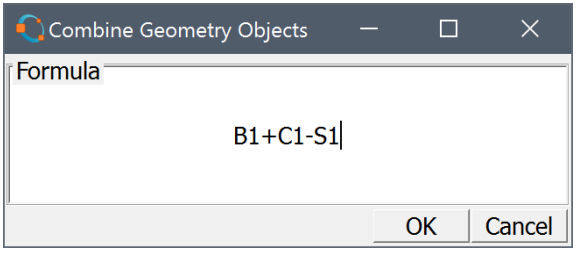
- Subtract objects from the largest of the selected geometry objects.



- Intersection of geometry objects.

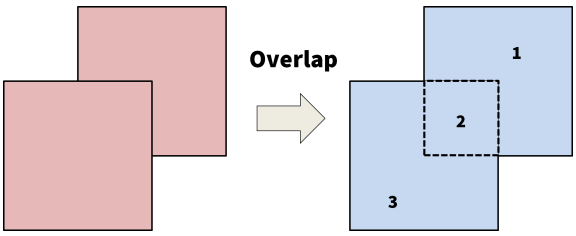


Alternatively to these buttons, the **Combine Objects...** geometry menu option opens a dialog box where an exact geometry formula can be entered. The syntax uses + for join, - for subtract, and & for the intersection operation as well as the geometry object labels or tags shown in the geometry object *Selection* listbox.



Note that if more than one geometry object is present in the final geometry FEATool will split it along all intersection borders to create subdomains as in the illustration below. The different subdomains can be used to set different material parameters, equation coefficients,

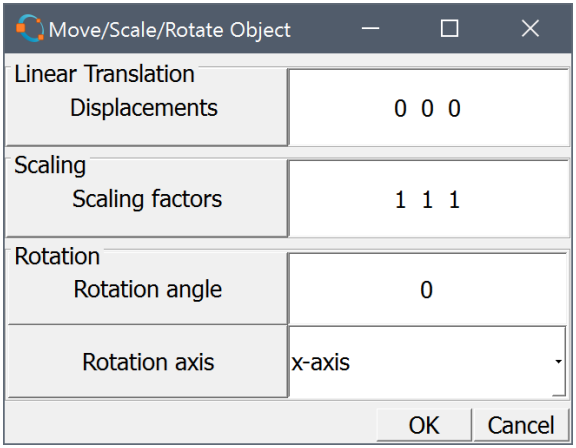
and boundary conditions along the separations. See for example the heat transfer model Shrink Fitting of an Assembly.



Also, the three dimensional geometry engine is not quite robust enough to handle all cases of overlapping boundary faces. Thus it is recommended when subtracting objects in 3D to allow for this by creating subtraction objects larger than the desired domain (so that non parallel planes are avoided).

The undo/split button is used to undo a previous *join*, *subtract*, *intersect*, *move*, *scale*, or *rotate* operation and returns the corresponding root objects.

In addition, the button allows one to rotate, scale, and move (linear translation) of a selected geometry object. Movement is specified as a space separated vector of a translation length in each space dimension (can be negative). Scaling, is similarly specified with a vector where a negative value will mirror the object. The rotation angle is specified in degrees (and in 3D the rotation axis must also be selected). Note that the default values will skip an operation (scaling of 1, translation and rotation 0).



Lastly, the delete button completely removes selected geometry objects.

2.5 Examples

The examples in the tutorial sections give step by step guides how to create different geometries. For example the Thin Plate with Hole model shows how to make a rectangle with a circular corner and Shrink Fitting of an Assembly describes how to create a complex assembly with several subdomains.

2.6 Command Reference

The following Matlab and Octave functions can be used on the command line to generate and modify objects and geometries.

Function	Description
gobj_block	Create a 3D block
gobj_circle	Create a circle in 2D
gobj_cylinder	Create a cylinder in 3D

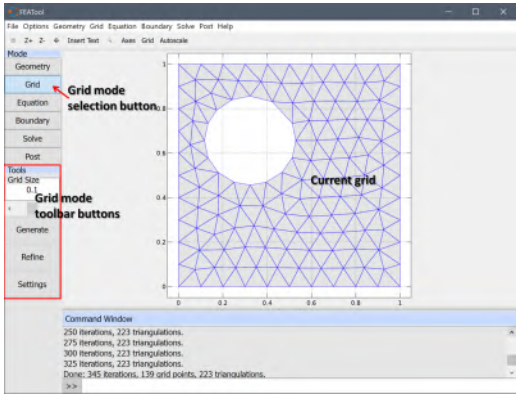
gobj_ellipse	Create an ellipse in 2D
gobj_line	Create a 1D line
gobj_point	Create a point
gobj_polygon	Create a 2D polygon
gobj_rectangle	Create a 2D rectangle
gobj_sphere	Create a 3D sphere
geom_apply_formula	Apply formula to geometry objects
geom_apply_transformation	Apply transformation to geometry objects
geom_split_object	Split composite geometry object
plotgeom	Plot and visualize geometry
impexp_gmsh	STL CAD geometry import and export
impexp_stl	Gmsh (geo) geometry export
impexp_triangle	Triangle (poly) geometry export

3 Grid

After a model geometry has been defined, a computational grid or mesh must be generated to allow for the finite element discretization. This section describes how to create or import a suitable grid.

3.1 Grid Mode

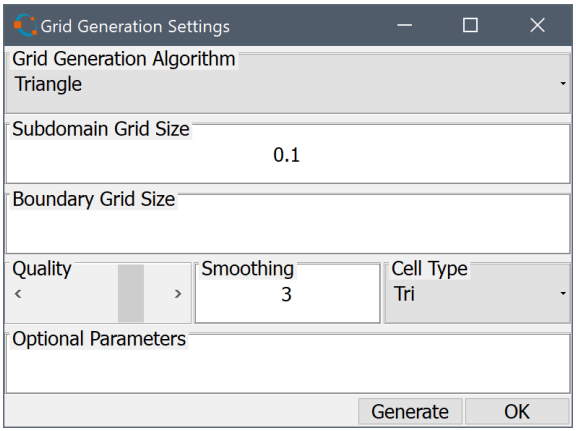
Grid mode can be selected by pressing the Grid mode button or corresponding menu option. In grid mode the toolbar buttons allow for grid generation, refinement, and accessing the advanced grid generation settings.



- Generate calls the selected grid generation function to generate and unstructured. The desired overall mean **Grid Size** is specified in the corresponding edit field which can be set manually or with the slider control.

- The Refine button is used to uniformly refine a grid. This will split triangular and quadrilateral (2D) grid cells into four subcells, and tetrahedral and hexahedral (3D) ones into eight subcells each. Grid points on boundaries will also be aligned with any geometries in 2D.

- Advanced grid settings can be controlled by pressing the Settings button which opens the **Grid Generation Settings** dialog box



The grid settings allows selecting the **Grid Generation Algorithm**, either the built-in *DistMesh* algorithm, or the external grid generators *Triangle* (only in 2D) and *Gmsh*. The **Subdomain Grid Size** and **Boundary Grid Size** edit fields can be used to prescribe maximum grid sizes for subdomains and boundaries. The **Quality** slider allows setting the target mesh quality (supported by *DistMesh* and *Triangle*). The number of grid **Smoothing** steps can be specified in the corresponding edit field. **Cell Type** allows selecting between unstructured triangular or structured quadrilateral grid cells (supported by *DistMesh* and *Gmsh*). And lastly, the **Optional Parameters** fields allows for an additional string of property/value pairs to be supplied to the grid generation call.

The **Grid** menu options allows for accessing and specifying the **Grid Generation Call...** which is a Matlab/Octave function call to support external grid generation tools instead of the default one. **Convert Grid Cells** is used to convert between triangular and quadrilateral cells in 2D, and between tetrahedral and hexahedral cells in 3D.

The **Grid** menu also supports the following import and export of grids through external ASCII format files

- **Import Grid** > **FEniCS/Dolfin Format...** imports a grid in `FEniCS/Dolfin` format (.xml file) (calls impexp_dolfin).
- **Import Grid** > **GiD Format...** imports a grid in `GiD` format (.msh file) (calls impexp_gid).
- **Import Grid** > **Gmsh Format...** imports a grid in `Gmsh` format (.msh file) (calls impexp_gmsh).
- **Import Grid** > **GMV Format...** imports a grid in `General Mesh Viewer` (GMV) format (.gmv file) (calls impexp_gmv).
- **Import Grid** > **Triangle Format...** imports a 2D `Triangle` grid (.node/.ele files) (calls impexp_triangle).

- **Export Grid** > **FEniCS/Dolfin Format...** exports a grid in `FEniCS/Dolfin` format (.xml file) (calls impexp_dolfin).
- **Export Grid** > **GiD Format...** exports a grid in `GiD` format (.msh file) (calls impexp_gid).
- **Export Grid** > **Gmsh Format...** exports a grid in `Gmsh` format (.msh file) (calls impexp_gmsh).
- **Export Grid** > **GMV Format...** exports a grid in `General Mesh Viewer` (GMV) format (.gmv file) (calls impexp_gmv).
- **Export Grid** > **OpenFOAM Format...** exports a grid in `OpenFOAM` format (calls grid2foam).
- **Export Grid** > **Triangle Format...** exports a 2D `Triangle` polygonal geometry description (.poly file) (calls impexp_triangle).

Note that in one dimension (1D) the only available option is to create a line grid.

3.2 External Grid Generators

FEATool Multiphysics comes with built-in support for the external mesh generators Gmsh and Triangle. Using either *Gmsh* or *Triangle* typically allows for faster and more robust grid generation, and also supports specifying different grid sizes on subdomains and boundaries. The *Grid Generation Algorithm* can be selected with the corresponding option in the **Grid Generation Settings** dialog box.

3.2.1 Gmsh and Triangle Installation

If FEATool cannot find the Gmsh or Triangle binaries, FEATool will try to automatically download, (compile), and install them. To install them manually, download *Gmsh* from http://gmsh.info/, and *Triangle* from http://www.cs.cmu.edu/~quake/triangle.html, then extract (compile) and install the binaries into the

```
lib/gmsh/
lib/triangle/
```

subdirectories of the FEATool installation directory.

3.2.2 Usage Notes

- Gmsh and Triangle can be used in the FEATool GUI as well as on the Matlab and Octave command lines with the gridgen_gmsh and gridgen_triangle commands.
- External grid generators works by exporting the current FEATool geometry in its own native format, make a system call to the grid generator binary, and then re-import the generated grid.
- For geometries with multiple and overlapping geometry objects the actual subdomain numbering will generally not correspond to the geometry object numbering (two inter-

secting geometry objects will for example create three or more subdomains and several internal boundaries). In this case the actual subdomain and boundary numbering for vector valued *hmax* and *hmaxb* arrays can easiest be visualized and determined by first creating a coarse grid and switching to *Equation/Subdomain* and *Boundary* modes, respectively.

- *Gmsh* propagates the *hmax* and *hmaxb* values down to the specific nodes in the mesh which means that it is currently not possible to exactly define mesh sizes for subdomains and boundaries.

3.3 Grid Reference Material

This section describes the format of the grid data structure that FEATool employs as well as advanced command line (CLI) usage such as manually creating and importing grids.

3.3.1 Grid Format

The grid format used by FEATool is specified in the **grid** struct with the following fields

Field	Description	Size
p	Grid point coordinates	(n_sdim, n_points)
c	Grid cell connectivity	(n_edges_per_cell, n_cells)
a	Grid cell adjacency	(n_edges_per_cell, n_cells)
b	Boundary information	(3+n_sdim, n_boundary_points)
s	Grid cell subdomain list	(1, n_cells)

The coordinates of the grid points are specified by an array **p**, with the row number indicating the *i*-th coordinate direction, and column number the corresponding grid point number.

Cell connectivities are specified in the **c** array, where each column point to the grid points (in **p**) making up each cell. The row index gives the local vertex number and the column index

the cell number. Moreover, the grid points are numbered counterclockwise in each cell.

Adjacency, meaning pointers to neighboring cells, are specified in the **a** array. Similar to **c** the row index gives the local edge or face number (starting with the corresponding grid point in **c**) and the column index points to the cell number. If the edge is on an external boundary the corresponding entry in **a** is *0*.

Boundary information is specified in the **b** array. The cell, edge/face, and boundary numbers are prescribed in the first to third rows. The last *n_sdim* rows consists of the outward pointing unit normals corresponding to each boundary cell edge (or face in 3D).

A list of subdomain numbers for each cell is given in the **s** vector.

3.3.2 Grid Cell Definitions

The various grid cells available in FEATool are defined in this sub section.

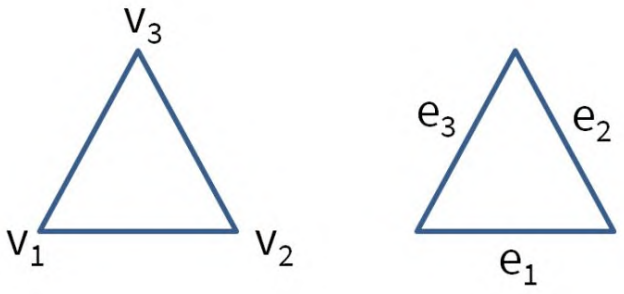
1D Line

In one dimension only the simple straight line segment grid cell is available. Grid Example 1 shows how this can be defined and used.

2D Triangle (Simplex)

The 2D triangular grid cell is defined with vertices in counter clockwise order. Local edges e_i are defined by the local vertices v_i as

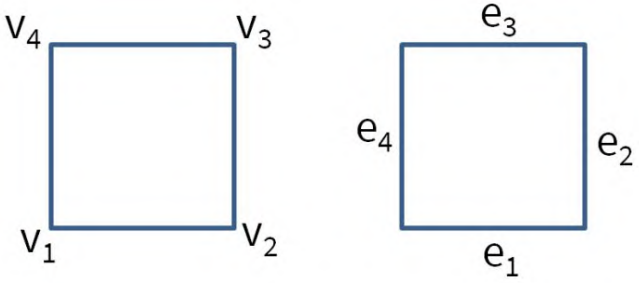
```
[ e1 ;     [ v1 v2 ;
  e2 ; =     v2 v3 ;
  e3 ]       v3 v1 ];
```



2D Quadrilateral

The quadrilateral grid cell is similarly defined with vertices in counter clockwise order. Local edges e_i are defined by the local vertices v_i as

```
[ e1 ;     [ v1 v2 ;
  e2 ;       v2 v3 ;
  e3 ] =     v3 v4 ;
  e4 ]       v4 v1 ];
```



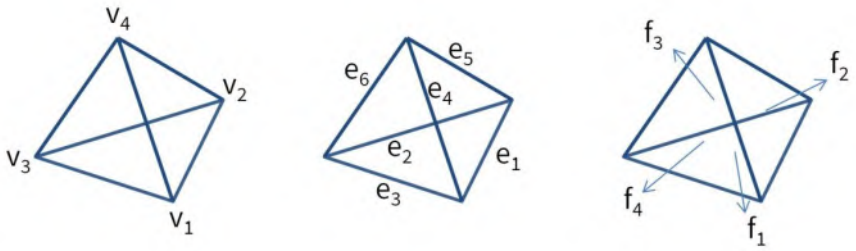
3D Tetrahedron (Simplex)

The 3D tetrahedral grid cell is defined with vertices in counter clockwise order. Local edges e_i are defined by the local vertices v_i as

```
[ e1 ;     [ v1 v2 ;
  e2 ;       v2 v3 ;
  e3 ;       v3 v1 ;
  e4 ; =     v1 v4 ;
  e5 ;       v2 v4 ;
  e6 ]       v3 v4 ];
```

The local faces f_i are defined by the local vertices v_i as

```
[ f1 ;     [ v1 v2 v3 ;
  f2 ;       v1 v2 v4 ;
  f3 ; =     v2 v3 v4 ;
  f4 ]       v3 v1 v4 ];
```



3D Hexahedron

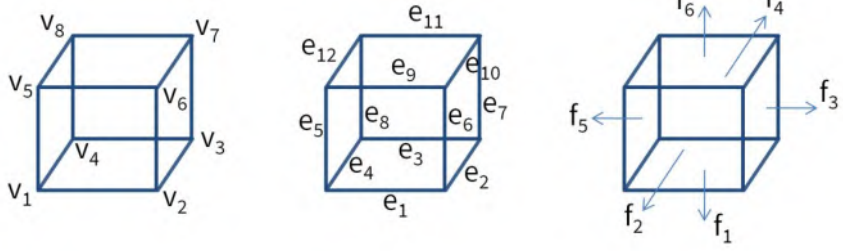
The hexahedral brick grid cell is defined with vertices in counter clockwise order. Local edges e_i are defined by the local vertices v_i as

```
[ e1  ;     [ v1 v2 ;
  e2  ;       v2 v3 ;
```

```
  e3  ;       v3 v4 ;
  e4  ;       v4 v1 ;
  e5  ;       v1 v5 ;
  e6  ;       v2 v6 ;
  e7  ; =     v3 v7 ;
  e8  ;       v4 v8 ;
  e9  ;       v5 v6 ;
  e10 ;       v6 v7 ;
  e11 ;       v7 v8 ;
  e12 ]       v8 v1 ];
```

The local faces f_i are defined by the local vertices v_i as

```
[ f1 ;     [ v1 v2 v3 v4 ;
  f2 ;       v1 v2 v6 v5 ;
  f3 ;       v2 v3 v7 v6 ;
  f4 ; =     v3 v4 v8 v7 ;
  f5 ;       v4 v1 v5 v8 ;
  f6 ]       v5 v6 v7 v8 ];
```



3.3.3 Grid Example 1

The following code can be used to manually define a one dimensional grid with 10 uniformly spaced cells on the line (0..1)

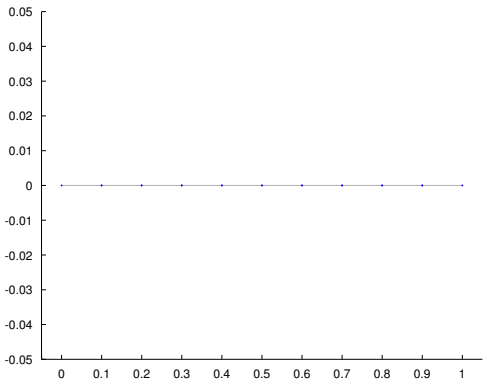
```
grid.p = 0:0.1:1;
grid.c = [1:10;2:11];
grid.a = [0:9;2:10 0];
grid.b = [1 10;1 2;1 2;-1 1];
grid.s = ones(1,10);
```

Alternatively one can use the grid utility function linegrid

```
grid = linegrid( 10, 0, 1 );
```

The plotgrid function can be used to plot and visualize a grid

```
plotgrid( grid )
```



3.3.4 Grid Example 2

A 2 by 2 unit square rectangular grid can be created with the following commands

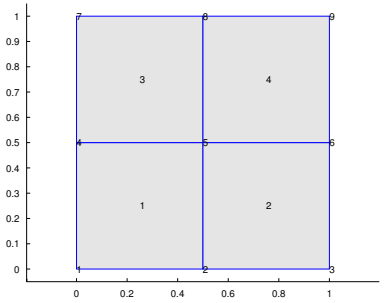
```
grid.p = [repmat([0,0.5,1],1,3);0 0 0 0.5 0.5 0.5 1 1 1];
grid.c = [1 2 5 4;2 3 6 5;4 5 8 7;5 6 9 8]';
grid.a = [0 2 3 0;0 0 4 1;1 4 0 0;2 0 0 3];
grid.b = [1 2 2 4 4 3 3 1;1 1 2 2 3 3 4 4;1 1 2 2 3 3 4 4; ...
          0 0 1 1 0 0 -1 -1;-1 -1 0 0 1 1 0 0];
grid.s = ones(1,4);
```

The rectgrid function can also be used to create rectangular grids, in this case

```
grid = rectgrid( 2, 2, [0 1;0 1] );
```

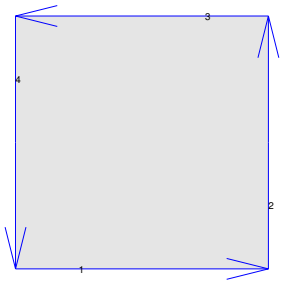
As before the grid can be plotted, visualizing both grid point and cell numbers, with

```
plotgrid( grid, 'nodelabels', 'on', 'cellabels', 'on' )
```



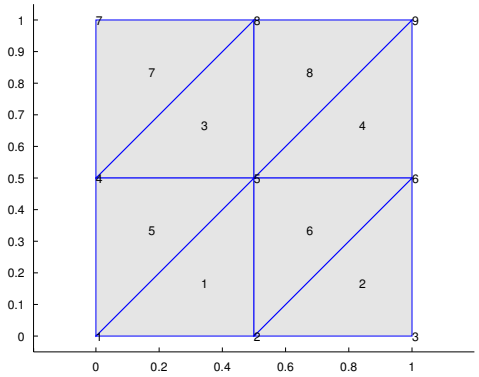
Similarly, the boundaries can be visualized with the function plotbdr (subdomains can be visualized with plotsubd)

```
plotbdr( grid )
```



As FEATool also supports simplex triangular cells in two dimensions a grid consisting of quadrilaterals can easily be converted with the utility function quad2tri

```
grid = quad2tri( grid )
```



Reverse conversions can be made with tri2quad function which uses Catmull-Clark subdivision. In 3D the hex2tet and tet2hex functions perform similar conversions.

3.3.5 Grid Example 3

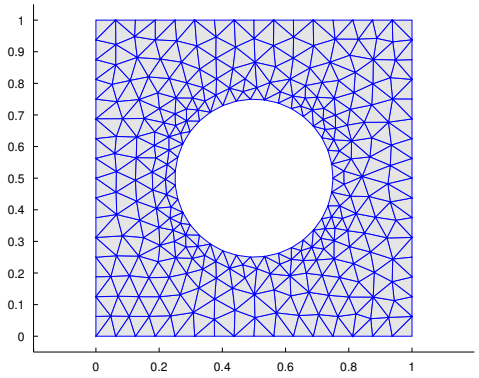
A more complex example, a grid for a rectangle with a circular hole can be created by first creating geometry objects (a rectangle and a circle), applying a formula to construct the geometry shape, and then calling the automatic unstructured grid generation function gridgen

```
% Geometry definition.
xmin  = 0;
xmax  = 1;
ymin  = 0;
ymax  = 1;
tag1  = 'R1';
gobj1 =  gobj_rectangle( xmin, xmax, ymin, ymax, tag1 );

xc    = (xmax-xmin)/2;
yc    = (ymax-ymin)/2;
r     = 0.25;
tag2  = 'C1';
gobj2 =  gobj_circle( [xc yc], r, tag2 );
```

```
geom.objects = { gobj1 gobj2 };
formula = 'R1 - C1';
geom = geom_apply_formula( geom, formula );
fea.geom = geom;

% Grid generation.
hmax = 0.1;
fea.grid = gridgen( fea, 'hmax', hmax );
```

As before the grid can be plotted, visualizing both grid point and cell numbers, with

```
plotgrid( grid, 'nodelabels', 'on', 'cellabels', 'on' )
```



3.3.6 Grid Import and Export

FEATool allows importing and exporting finite element grids between FEniCS, GiD, Gmsh, General Mesh Viewer (GMV), OpenFOAM, and Triangle formats with the correspond-

ing impexp_dolfin, impexp_gid, impexp_gmsh, impexp_gmv, impexp_foam, and impexp_triangle functions. These functions can also be accessed from the **Grid** and **Postprocessing** menus of the FEATool GUI.

Due to the simple grid format syntax it is possible to manually import grids from other software. The process essentially consists of first exporting the grid point coordinates and grid cell connectivity data from the external grid generation tool into separate text files. Then import them into Octave or Matlab, after which they can be reshaped and used by FEATool. The following steps describe the process

1. The first step is to output the grid to an ASCII text format. If possible save the grid output to two files, one for the grid vertex coordinates, and another with the grid cell connectivities (this specifies which grid points/vertices make up each cell). If this is not possible one will have to manually open the grid output file in a text editor, and cut and save the grid coordinates and cell connectivities to two different files.

2. Load the two files in Octave/Matlab (here it is assumed they are saved as *p.txt* and *c.txt*):

```
load p.txt
load c.txt
```

3. Reshape the grid coordinates (**p** variable) so that it has the form $n_{sdim} \times n_p$ where n_{sdim} is the number of space dimensions (1, 2, or 3) and n_p is the total number of grid points (**p** essentially consists of rows of x, y, and z-coordinates). Precisely how to reshape depends on the output format from the external grid generator, one might not have to do anything (check the shape by entering the command **size(p)** or **whos**), it might be enough to transpose as **p = p';**, or one might have to reshape with something like **p = reshape(p,n_sdim,n_p);**.

4. Similarly, reshape the cell connectivity array **c** so that it has the shape $n_v \times n_c$ where n_v is the number of vertices on each cell (for example 3 for triangles) and n_c is the total number

of cells. Each column should point to the corresponding grid points in **p** that make up the cell.

5. The grid vertices must run in counterclockwise order on each cell, so reorient them if necessary. This is usually accomplished by changing the row order, for example **c = c([3 2 1],:);** which the ordering for triangles.

6. Use the function gridadj to create an array that points to neighboring cells for each cell edge

```
n_sdim = size(p,1);
a = gridadj( c, n_sdim );
```

7. Create a vector that assigns a subdomain number for each cell. If the geometry should be one single block and thus not split, a unit row vector is sufficient

```
n_c = size(c,2);
s = ones( 1, n_c );
```

8. Use the function gridbdr to create boundary edge/face information. The *gridbdr* function splits external boundaries where the angles are acute

```
b = gridbdr( p, c, a );
```

9. Alternatively, one can manually construct and set the boundary numbering, for example as

```
[ix_edge_face,ix_cells] = find( a==0 );
b = [ix_cells'; ix_edge_face'; ones(1,length(ix_cells))];
b = [b; gridnormals(p,c,b)];
```

this creates a boundary array **b** where all external boundaries are joined. With additional boundary information one can now split the boundary edges and faces (columns) by assigning different boundary numbers (third row in **b**). A helpful function to use here is findbdr which allows one to find boundary numbers and column indices in to **b** by prescribing logical expressions such as *x>0.5*. For example

```
[~,ix] = findbdr( struct('p',p,'c',c,'b',b), 'x>0.5', false );
b(3,ix) = 2;
```

sets boundary number *2* on all boundary edges/faces where *x>0.5* is true.

10. Create a grid *struct* to hold all the grid information

```
grid.p = p;
grid.c = c;
grid.a = a;
grid.s = s;
grid.b = b;
```

11. Now the grid can be used by FEATool functions and subroutines on the command line.

12. Optionally, one can also import the grid into the FEATool GUI by using the Menu option

```
File > Import > Variables From Main Workspace...
```

Select the created **grid** variable to import and press **Import**. This loads it into the local FEATool memory workspace and can now be accessed from the FEATool Terminal (the bottom command $**>>**$ edit field in the main GUI window). By entering the command

```
fea.grid = grid;
```

in the FEATool Terminal command line (not the main Octave/Matlab command window) effectively replaces the current grid with the imported one. Press the **Grid** mode button to update and show the new grid.

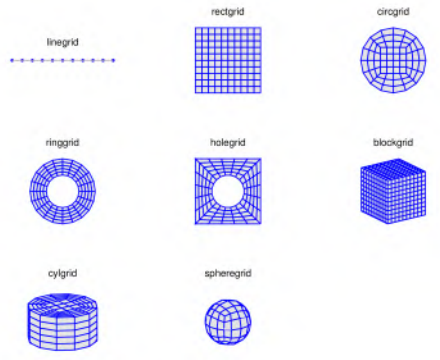
3.3.7 Creating Structured Grids

The computational finite element library in FEATool supports FEM shape functions for structured grids (quadrilaterals in 2D and hexahedra in 3D). Although more difficult to generate automatically, structured grids are often computationally superior, allowing for higher accuracy with a smaller number of cells.

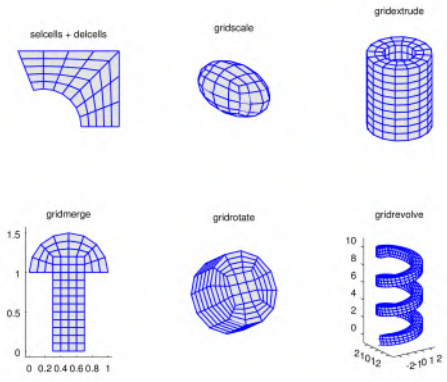
In FEATool, structured tensor-product grids of basic geometric shapes can easily be generated on the command line with the following Octave/Matlab m-script functions

Function	Description
linegrid	Create a 1D line grid
circgrid	Create a 2D structured circular grid of quadrilateral cells

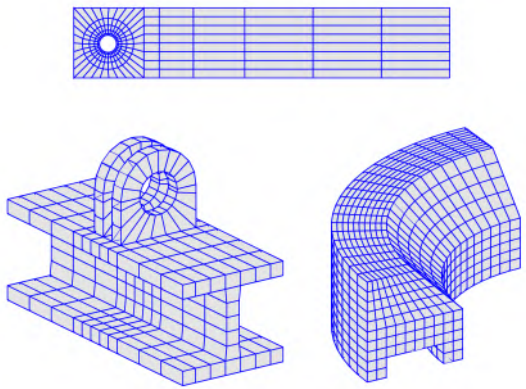
holegrid	Create a 2D rectangular grid with a circular hole
rectgrid	Create a 2D rectangular grid of quadrilateral cells
ringgrid	Create a 2D grid of a ring shaped domain
blockgrid	Create a 3D structured block grid
cylgrid	Create a 3D structured cylindrical grid
spheregrid	Create a 3D grid for a spherical domain



Moreover, the grid utility functions selcells, delcells, gridextrude, gridmerge, gridrevolve, gridrotate, and gridscale can be used to manually modify, transform, and join grids to more complex structures.



The figure below shows three examples of more complex grids created by using these functions.



The first flow over cylinder benchmark grid is for example created with the following commands:

```
grid1 = ringgrid( [0.05 0.06 0.08 0.11 0.15], ...
                  32, [], [], [0.2;0.2] );
grid2 = holegrid( 8, 1, [0 0.41;0 0.41], 0.15, [0.2;0.2] );
```

```
grid2 = gridmerge( grid1, 5:8, grid2, 1:4 );
grid3 = rectgrid( [0.41 0.5 0.7 1 1.4 1.8 2.2], ...
                  8, [0.41 2.2;0 0.41] );
grid  = gridmerge( grid3, 4, grid2, 6 );
```

And the lower right revolved grid with these commands:

```
grid = rectgrid();
grid = gridscale( grid, {'1-(y>0.5).*(y-0.5)' 1} );
grid = delcells( selcells( ...
         '((x<=0.8).*(x>=0.2)).*(y<=0.2)', grid ), grid );
grid = gridrevolve( grid, 20, 0, 1/4, 2, pi/2, 0 );
```

The last example with two brackets attached to an I-beam section is more complex:

```
grid01 = ringgrid( 1, 20, 0.03, 0.06, [0;0] );
indc01 = selcells( grid01, 'y<=sqrt(eps)' );
grid01 = delcells( grid01, indc01 );

grid02 = holegrid( 5, 1, .06*[-1 1;-1 1], .03, [0;0] );
indc02 = selcells( grid02, 'y>=-sqrt(eps)' );
grid02 = delcells( grid02, indc02 );
grid2d = gridmerge( grid01, [5 6], grid02, [7 8] );

grid1 = gridextrude( grid2d, 1, 0.02 );
grid1 = gridrotate( grid1, pi/2, 1 );
grid2 = grid1;
grid1.p(2,:) = grid1.p(2,:) + 0.03;
grid2.p(2,:) = grid2.p(2,:) - 0.01;

x_coord = [ -0.08 linspace(-0.06,0.06,6) 0.08 ];
y_coord = [ -0.2 -0.15 -0.1 -0.05 -0.03 -0.01 ...
             0.01  0.03  0.05  0.1  0.15  0.2 ];
grid3 = blockgrid( x_coord, y_coord, 1, ...
                   [-0.08 0.08;-0.2 0.2;-0.08 -0.06] );
grid4 = blockgrid( 1, y_coord, 5, ...
                   [-0.01 0.01;-0.2 0.2;-0.18 -0.08] );
grid5 = grid3;
```

```
grid5.p(3,:) = grid5.p(3,:) - 0.12;

grid = gridmerge( grid1, 9, grid3, 6 );
grid = gridmerge( grid2, 9, grid, 17 );
grid = gridmerge( grid4, 6, grid, 23, 1 );
grid = gridmerge( grid5, 6, grid, 29, 2 );
```

After, the grids have been created on the command line they can also be imported into the FEATool GUI (by using the **File** > **Import** > **Variables From Main Workspace option**, after which the command **fea.grid = grid;** needs to be entered at the FEATool command prompt).

3.3.8 Quadrilateral Grid Generation

Using quadrilateral grid cells are often advantageous to simplex and triangular cells in that they can provide somewhat more accuracy when aligned with geometry features and also tends to require less overall grid cells. Quadrilateral grid generation can be specified by selecting *Quad* grid **Cell Type** option in the Grid Settings dialog box (available with *DistMesh* and *Gmsh* in 2D). Quadrilaterals unfortunately do not tend to allow for easy automatic grid generation although It is also possible to subdivide triangles and convert into quads the resulting grids are often of poor quality.

The optional gridgen_quad command line function was originally designed to align quadrilateral cell edges with immersed interfaces described by distance and level set functions (analog to DistMesh). The algorithm then uses the zero level set contour from the distance functions to align grid cell edges with external geometry object boundaries. Furthermore, the cells are split in a way to treat edge cases such as when an interface segment crosses a cell diagonal.

3.3.9 Grid Utility Functions

The following functions are available for working with and processing grids

Function	Description
gridgen	Unstructured automatic grid simplex generation (DistMesh)
gridgen_gmsh	External grid generation with Gmsh (2D and 3D)
gridgen_triangle	External grid generation with Triangle (2D)
gridgen_quad	Unstructured automatic grid quadrilateral generation
gridrefine	Refine a grid uniformly
gridsmooth	Apply smoothing to a grid
gridextrude	Extrude a grid from 2D to 3D
gridrevolve	Extrude and revolve a 2D grid to 3D
gridrotate	Rotate a grid along a specified axis
gridscale	Apply scaling to a grid
gridmerge	Merge two grids along boundaries
quad2tri	Convert a grid of quadrilateral cells to triangular cells
tri2quad	Convert a grid of triangular cells to quadrilateral cells
hex2tet	Convert a grid of hexahedral cells to tetrahedral cells
tet2hex	Convert a grid of tetrahedral cells to hexahedral cells
impexp_dolfin	Import and export FEniCS/Dolfin grid data
impexp_foam	Import and export OpenFOAM grid data
impexp_gid	Import and export GiD grid data
impexp_gmv	Import and export GMV grid and postprocessing data
impexp_gmsh	Import Gmsh grid and postprocessing data
impexp_triangle	Import and export 2D Triangle grid and polygon data

gridcheck	Check grid for errors
gridadj	Create grid adjacency information
gridbdr	Create grid boundary information
gridbdrx	Reconstruct internal boundaries
gridvert	Create grid vertex information
gridedge	Create grid edge information
gridface	Create grid face information
findbdr	Find boundary indices and numbers
assign_bdr	Sort boundaries according to geometry objects
project_bdrp	Project boundary points to geometric boundary
selcells	Find cell indices from an expression
delcells	Delete a selection of cells from a grid
plotbdr	Plot and visualize boundaries
plotsubd	Plot and visualize subdomains
plotgrid	Plot and visualize grid

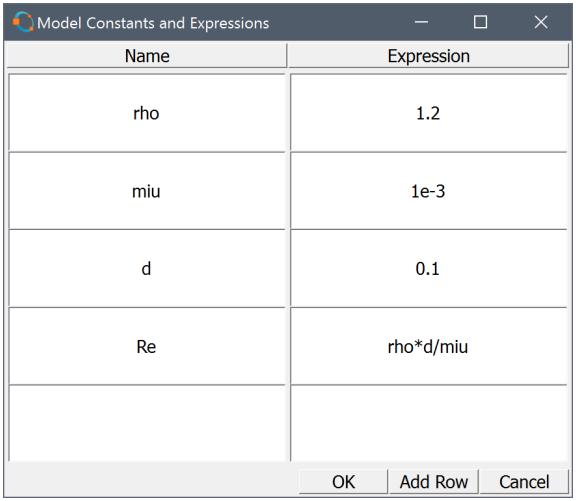
4 Physics

In **Equation** mode partial differential equations (PDEs) together with equation coefficients must be chosen to accurately describe the physical phenomena to be simulated. Furthermore, in **Boundary** mode suitable boundary conditions must be prescribed in order to account for how the model interacts with its surroundings (outside of the modeled geometry and grid).

4.1 Equation and Boundary Modes

After creating a suitable geometry and grid one can switch between equation and boundary modes by using the Equation and Boundary mode buttons. The corresponding **Equation** and **Boundary Settings** dialog boxes will also automatically open. Each boundary and equation setting tab corresponds to a different physics mode present in the model and allow for specifying the equation and boundary coefficients, initial conditions, and finite element shape functions. The coefficients and boundary coefficients will vary depending on the chosen physics mode and are explained in the following sections. The equation settings also allow for different equation and initial conditions to be set on a per subdomain basis by selecting the subdomains in the left hand side **Subdomain:** listbox.

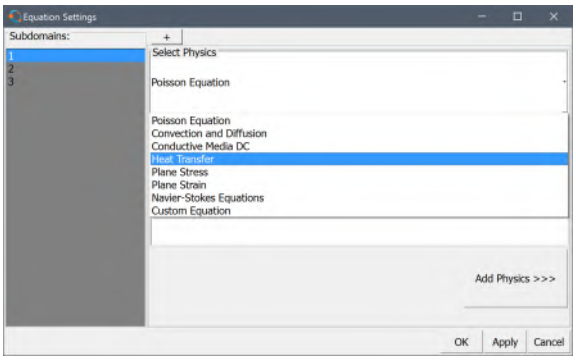
Moreover, a convenient way to set up models is to use **Model Constants and Expressions**. The button opens the corresponding dialog box where one can enter constants and define expressions. One entered they can be used in stead of entering numerical values in equation coefficients and postprocessing expressions. More space for coefficients and expressions can also be added with the **Add Row** button.



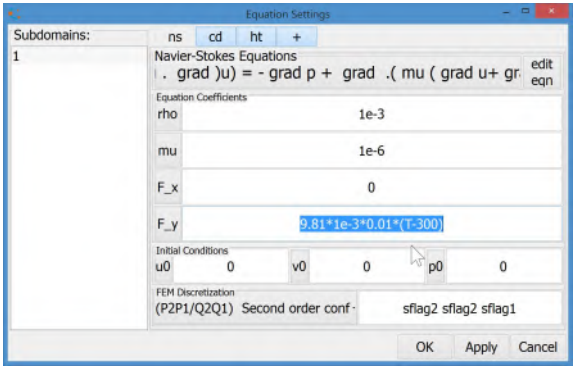
4.2 Multiphysics

The FEATool GUI also makes it easy to add and couple multiphysics equations and complex expressions to your models.

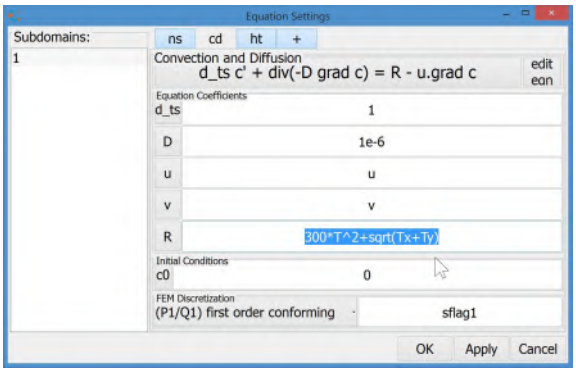
After starting to work with a physics mode it is simple to add one or more other modes by going to the last tab, with a **+** plus sign, in the **Equation Settings** dialog box. There you can simply choose an additional physics mode from the drop down combo box, select the dependent variable names you want to use (or keep the default ones), and press **Add Physics** >>> to add the mode. The new physics mode will now show up as a new tab with its short abbreviated name on the tab handle. Once the new mode has been added you can switch between the modes by clicking on the corresponding tabs in the Equation and Boundary Settings dialog boxes.



The physics modes can be coupled by simply using the dependent variable names and derivatives in the coefficient expression dialog boxes. For example, the Navier-Stokes equations physics mode shown below uses the temperature variable **T** from the heat transfer mode in the source term for the y-direction.



Moreover, here below we can see how to make a three way multiphysics coupling. The convective velocities **u** and **v** are coupled from the Navier-Stokes equations physics mode and at the same time the temperature **T** and its two derivatives **Tx** and **Ty** are simultaneously coupled to the reaction rate source term in the convection and diffusion mode.



As we have seen, it is very simple to set up multiphysics models in FEATool. This is made possible by the expression parsing functionality that allows you to enter and use complex expressions of dependent variables (for example **u**, **v**, **T**, **c**), their first derivatives (by just appending x or y to the variable names like **Tx** and **Ty**), the space dimensions **x** and **y**, as well as all common Octave and Matlab expressions and constants like **pi**, **sin**, **cos**, **sqrt**, **^2** etc.

4.3 Physics Modes

By using the predefined FEATool physics modes it is possible to easily and quickly implement models which simulate different physical effects such as fluid flow, structural stresses, chemical reactions, and heat transfer. This section describes how the various physics modes are defined, set up and used. The available physics modes are listed in the following table

Physics Mode	Description	Definition Function
Poisson Equation	Poisson equation	poisson
Convection and Diffusion	Mass transport through convection and diffusion	convectiondiffusion
Conductive Media DC	Electric potential	conductivemediadc
Electrostatics	Electrostatics	electrostatics

Magnetostatics	Magnetostatics	magnetostatics
Heat Transfer	Heat transport through convection and conduction	heattransfer
Euler-Bernoulli Beam	Euler-Bernoulli beam theory (1D)	eulerbeam
Plane Stress	Structural mechanics (2D plane stress approximation)	planestress
Plane Strain	Structural mechanics (2D plane strain approximation)	planestrain
Axisymmetric Stress-Strain	Structural mechanics (2D cylindrical coordinates)	axistressstrain
Linear Elasticity	Structural mechanics (3D solid linear elasticity)	linearelasticity
Darcy's Law	Darcy's law porous media flow	darcyslaw
Brinkman Equations	Brinkman equations porous media flow	brinkmaneqns
Navier-Stokes Equations	Incompressible fluid flow	navierstokes
Custom Equation	User defined equation	customeqn

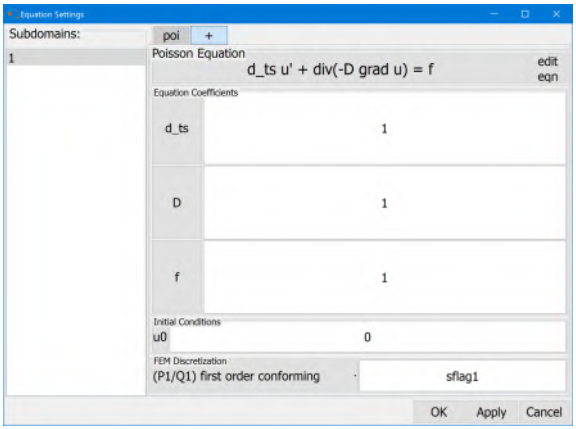
The equation parameters and coefficients for each selected physics mode are defined in the corresponding *Equation Settings* dialog boxes described below. In addition to also displaying the PDE equation, it is possible to change or edit the equation definitions with the **edit** equation button, and activate/deactivate physics modes in specific subdomains with the **active** button. Furthermore, the used finite element shape functions can also be selected either from the drop down box or directly entering a space separated list of shape functions for each dependent variable. Physics modes with convective effects allowing for numerical artificial stabilization also feature a button opening the stabilization settings.

4.3.1 Poisson Equation

The Poisson equation physics mode solves the classic elliptic Poisson equation for the scalar dependent variable u

$$d_{ts}\frac{\partial u}{\partial t} + \nabla \cdot (-D\nabla u) = f$$

where d_{ts} is a time scaling coefficient, D is a diffusion coefficient, and f is a scalar source term. In the Poisson *Equation Settings* dialog box shown below these equation coefficients, initial value $u_0 = u(t = 0)$ can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



The physics mode allows for both Dirichlet and Neumann (flux) boundary conditions. Dirichlet conditions prescribe a fixed value r of the dependent variable $u = r$ on a boundary segment, while a Neumann condition will prescribe the normal flux to a boundary segment, that is $-\mathbf{n} \cdot D\nabla u = g$, where $\mathbf{n}$ is the outward directed normal, and g therefore represents the value of the inward directed flux. The available boundary conditions are summarized in the table below

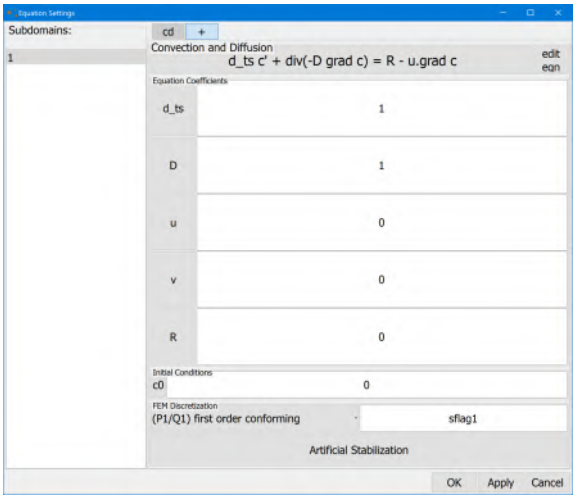
Boundary Condition	Definition	Boundary Coefficient
Dirichlet	Prescribed value, $u = r$	r
Neumann	Prescribed flux, $-\mathbf{n} \cdot D\nabla u = g$	g

4.3.2 Convection and Diffusion

The convection and diffusion physics mode models mass transport and reaction of a chemical species c. The governing equation for convection and diffusion reads

$$d_{ts}\frac{\partial c}{\partial t} + \nabla \cdot (-D\nabla c) = R - \mathbf{u} \cdot \nabla c$$

where d_{ts} is a time scaling coefficient, D is a diffusion coefficient, R is the reaction rate source term, and $\mathbf{u}$ a vector valued convective velocity field. In the *Equation Settings* dialog box shown below these equation coefficients, initial value $c_0 = c(t = 0)$ can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



For convection and diffusion problems with dominating convective effects it is advisable to use artificial stabilization. Pressing the lower **Artificial Stabilization** opens the corresponding dialog box which allows adding both isotropic artificial diffusion and anisotropic streamline diffusion. Turning coefficients are also provided to control the strength of the introduced artificial diffusion.

The convection and diffusion physics mode allows for four different boundary conditions; a prescribed concentration boundary condition, a convective flow (outflow condition), an insulation/symmetry condition which prescribes zero flux (or flow), and a prescribed flux condition.

These boundary conditions are summarized in the table below

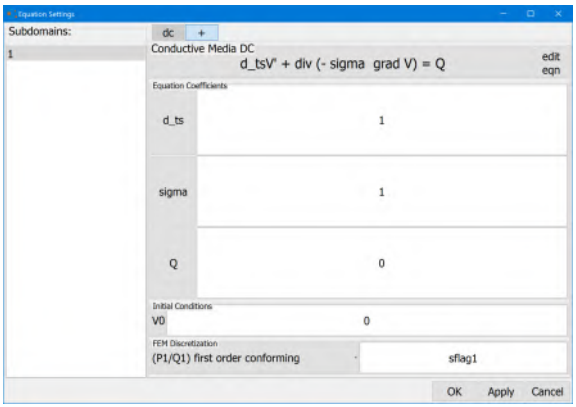
Boundary Condition	Definition	Boundary Coefficient
Concentration	Prescribed concentration, $c = c_0$	c_0
Convective flux	Outflow, $-\mathbf{n} \cdot D\nabla c = 0$	
Insulation/symmetry	Zero flux/flow, $-\mathbf{n} \cdot (D\nabla c + \mathbf{u}c) = 0$	
Flux condition	Prescribed flux, $-\mathbf{n} \cdot (D\nabla c + \mathbf{u}c) = N_0$	N_0

4.3.3 Conductive Media DC

Electric potential and current can be modeled through the conductive media DC physics mode. To describe the flow of current and electric potential V an electric field is firstly defined as $\mathbf{E} = \nabla V$. Furthermore, the current density $\mathbf{J}$ is related to the electric field by $\mathbf{J} = \sigma\mathbf{E}$ where σ is the conductivity. By assuming conservation of the current one can pose a continuity equation for the current density, that is $\nabla \cdot \mathbf{J} = Q$ where Q is a current source, which after expansion results in the following equation

$$d_{ts}\frac{\partial V}{\partial t} + \nabla \cdot (-\sigma\nabla V) = Q$$

In the *Equation Settings* dialog box shown below the equation coefficients, initial value for the electric potential $V_0 = V(t = 0)$ can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



The conductive media DC physics mode features two boundary conditions, one prescribing the electric potential at a boundary $V = V_0$, and the other prescribing the current flow into the boundary. These boundary conditions are summarized in the table below

Boundary Condition	Definition	Boundary Coefficient
Electric potential	Prescribed electric potential, $V = V_0$	V_0
Current flow	Prescribed current flow (flux), $-\mathbf{n} \cdot (\sigma \nabla V) = J_0$	J_0

4.3.4 Electrostatics

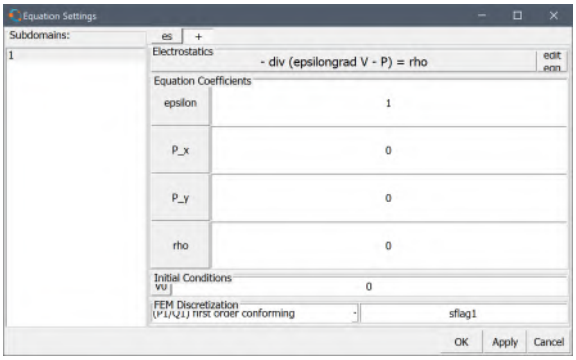
The electrostatics physics mode is an extension of the conductive media DC mode allowing for polarization vector, *P*, to be taken into account. The electrostatics equation for the electric potential *V* then reads

$$-\nabla \cdot (\varepsilon \nabla V - \mathbf{P}) = \rho$$

where ε is the permittivity and ρ the charge density source term.

In the *Equation Settings* dialog box shown below the equation coefficients, initial value for the electric potential $V_0 = V(t = 0)$ can be specified. The FEM shape function space can also be

selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



The electrostatics physics mode features two boundary conditions, one prescribing the electric potential at a boundary $V = V_0$, and the other prescribing the current flow into the boundary. These boundary conditions are summarized in the table below

Boundary Condition	Definition	Boundary Coefficient
Electric potential	Prescribed electric potential, $V = V_0$	V_0
Ground/antisymmetry	Zero electric potential, $V = 0$	
Surface charge	Prescribed current flow (flux), $-\mathbf{n} \cdot (-\varepsilon \nabla V + \mathbf{P}) = \rho_s$	rho_s
Insulation/symmetry	Zero current flow (flux), $-\mathbf{n} \cdot (-\varepsilon \nabla V + \mathbf{P}) = 0$	

4.3.5 Magnetostatics

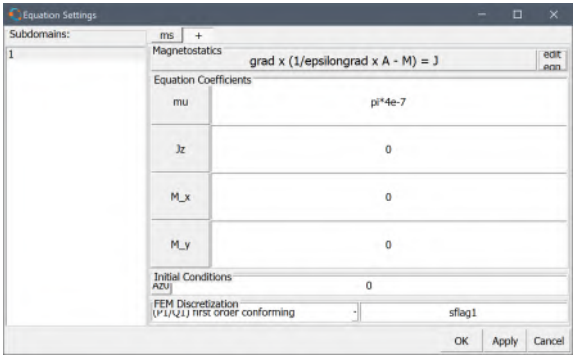
The magnetostatics physics mode models simplified Maxwell's equations and in two dimensions solves

$$-\nabla \cdot (\frac{1}{\mu}\nabla A_z - \mathbf{M}) = J_z$$

for the magnetic vector potential A_z in the z-direction where ε is the permeability, $\mathbf{M}$ the magnetization vector, and J_z the current density. In three dimensions the equations are simplified to to exclude currents which reduces them to solving for a scalar magnetic potential V_m

$$-\nabla \cdot (\mu \nabla V_m - \mu \mathbf{M}) = 0$$

In the *Equation Settings* dialog box shown below the equation coefficients, initial value for the magnetic potential can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



The magnetostatics physics mode features four boundary conditions, one prescribing the magnetic potential at a boundary, one for magnetic insulation zero potential, surface current, and magnetic insulation/symmetry. These boundary conditions are summarized in the table below

Boundary Condition	Definition	Boundary Coefficient
Magnetic potential	Prescribed magnetic potential, $A_z/V_m = A_{z,0}/V_{m,0}$	Az_0/Vm_0

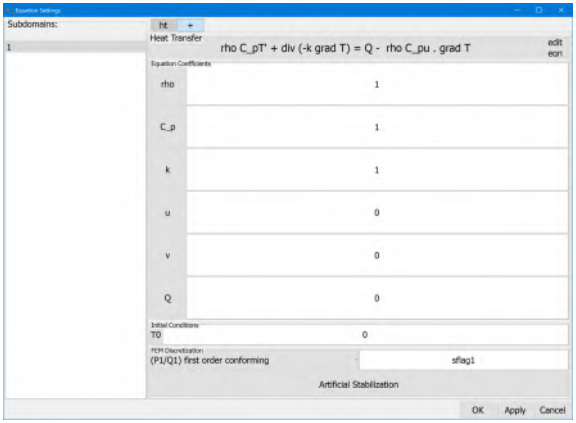
Magnetic insulation/antisymmetry	Zero magnetic potential, $A_z/V_m = 0$	
Surface current	Prescribed current flow (flux), $-\mathbf{n}\cdot(\frac{1}{\mu}\nabla A_z - \mathbf{M}/\mu\nabla V_m - \mu\mathbf{M}) = \mathbf{J}_s$	$J_{x/y/z,s}$
Electric insulation/symmetry	Zero current flow (flux), $-\mathbf{n}\cdot(\frac{1}{\mu}\nabla A_z - \mathbf{M}/\mu\nabla V_m - \mu\mathbf{M}) = 0$	

4.3.6 Heat Transfer (Convection and Conduction)

The heat transfer physics mode models heat transport through convection and conduction and heat generation through the following governing equation

$$\rho C_p \frac{\partial T}{\partial t} + \nabla\cdot(-k\nabla c) = Q - \rho C_p \mathbf{u}\cdot\nabla T$$

where ρ is the density, C_p the heat capacity, k is the thermal conductivity, Q is the heat source term, and $\mathbf{u}$ a vector valued convective velocity field. In the *Equation Settings* dialog box show below the equation coefficients, initial value for the temperature $T_0 = T(t=0)$ can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



For heat transfer problems with dominating convective effects it is advisable to use artificial stabilization. Pressing the lower **Artificial Stabilization** opens the corresponding dialog box which allows adding both isotropic artificial diffusion and anisotropic streamline diffusion. Turning coefficients are also provided to control the strength of the introduced artificial diffusion.

The heat transfer physics mode allows for four different boundary conditions; prescribed temperature, convective flow (outflow condition), an insulation/symmetry condition which prescribes zero flux (or flow), and a prescribed flux boundary condition.

The heat flux boundary condition involves several parameters. Firstly, an arbitrary expression for the heat flux may be prescribed with the coefficient $\mathbf{q_0}$. The second term, $\mathbf{h*(T_{inf}-T)}$, represents natural convection between the boundary and the surroundings. Here **h** is the convective heat transfer coefficient and $\mathbf{T_{inf}}$ a reference bulk temperature. The final term, $\mathbf{Const*(T_{amb}\hat{}4-T\hat{}4)}$, represents a radiation flux boundary condition where **Const** is the product between the emissivity of the boundary ε and the Stefan-Bolzmann constant σ, and $\mathbf{T_{amb}}$ is the surrounding ambient temperature.

The boundary conditions are summarized in the table below

Boundary Condition	Definition	Boundary Coefficient
Temperature	Prescribed temperature, $T = T_0$	T_0

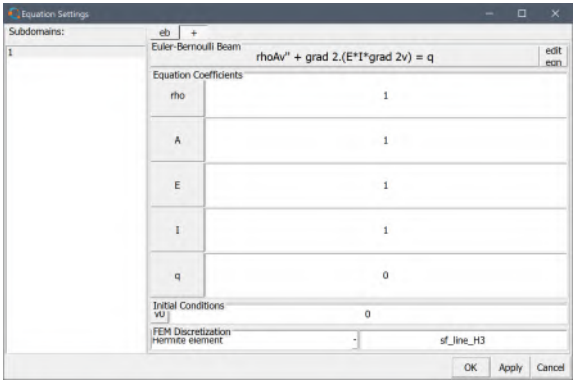
Convective flux	Outflow, $-\mathbf{n} \cdot k\nabla T = 0$	
Thermal insulation/symmetry	Zero heat flux, $-\mathbf{n} \cdot (k\nabla T + \rho C_p \mathbf{u} c) = 0$	
Heat flux	Prescribed heat flux, $-\mathbf{n} \cdot (k\nabla T + \rho C_p \mathbf{u} T) = q_0 + h \cdot (T_{inf} - T) + Const \cdot (T_{amb}^4 - T^4)$	q_0, h, T_{inf}, Const, T_{amb}

4.3.7 Euler-Bernoulli Beam

The Euler-Bernoulli beam physics mode models displacements, stresses and strains in a one-dimensional representations of beams and bars. The Euler-Bernoulli equations for the displacement v reads

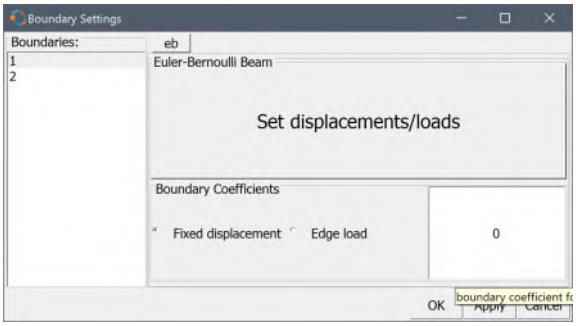
$$\rho A \frac{\partial^2 v}{\partial t^2} + \frac{\partial^2}{\partial x^2}(EI\frac{\partial^2 v}{\partial x^2}) = q$$

where ρ is the beam material density, A the cross-sectional area, E modulus of elasticity, I cross section moment of inertia, and q any distributed loads. In the Euler beam *Equation Settings* dialog box shown below these equation coefficients, initial value $v_0 = v(t = 0)$ can be specified. The FEM shape function space can also be selected from the drop-down combobox (Hermite C1 shape functions by default).



The boundary conditions for the Euler beam physics mode allows for any combination of prescribing displacements and edge loads. These boundary conditions are summarized in the table below.

Boundary Condition	Definition	Boundary Coefficient
Fixed displacements	Prescribed displacements, $u = u_0, v = v_0$	v_0
Edge loads	Prescribed edge loads, $F_x = f_{x,0}, F_y = f_{y,0}$	$f_{y,0}$



4.3.8 Plane Stress

The plane stress physics mode models how structural stresses form in thin structures where the planar component of the stress can be neglected or considered zero. In this case the stress-strain relations can be written as

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix} \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{pmatrix}$$

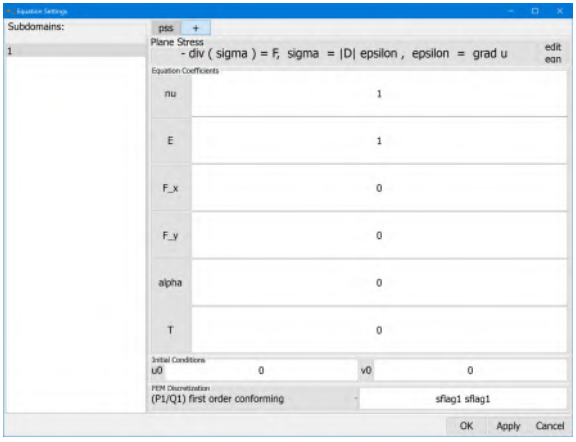
where E is the elastic or Young's modulus, and ν is the Poisson's ratio of the material. The strains are related to the material displacements (u, v) as

$$\varepsilon_x = \frac{\partial u}{\partial x}, \qquad \varepsilon_y = \frac{\partial v}{\partial y}, \qquad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$$

Balance equations for the stresses finally give the resulting governing equation system as

$$\begin{cases} -\dfrac{\partial \sigma_x}{\partial x} - \dfrac{\partial \tau_{xy}}{\partial y} = F_x \\ -\dfrac{\partial \tau_{xy}}{\partial x} - \dfrac{\partial \sigma_y}{\partial y} = F_y \end{cases}$$

where F_x and F_y are volume (body) forces in the x and y-directions, respectively. In the *Equation Settings* dialog box shown below the equation coefficients, initial value for the displacements can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



Optional stress-strain temperature dependence can also be added, which takes the form

$$\sigma = \mathbf{D}\varepsilon - \frac{E\alpha T}{1-\nu}[1 \;\; 1 \;\; 0]^T$$

where α is the coefficient of thermal expansion and T either a prescribed temperature field, a dependent variable name from another physics mode that represents the temperature, or a combination such as $T - T_{ref}$.

The boundary conditions for the plane stress physics mode allows for any combination of prescribing displacements and edge loads. These boundary conditions are summarized in the table below.

Boundary Condition	Definition	Boundary Coefficient
Fixed displacements	Prescribed displacements, $u = u_0, v = v_0$	u_0, v_0
Edge loads	Prescribed edge loads, $F_x = f_{x,0}, F_y = f_{y,0}$	$f_{x,0}$, $f_{y,0}$

4.3.9 Plane Strain

Like the plane stress physics mode, the plane strain mode models how structural stresses form in structures but where the z-component of the displacements can be neglected or considered zero. In this case the stress-strain relations can be written as

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{pmatrix} \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{pmatrix}$$

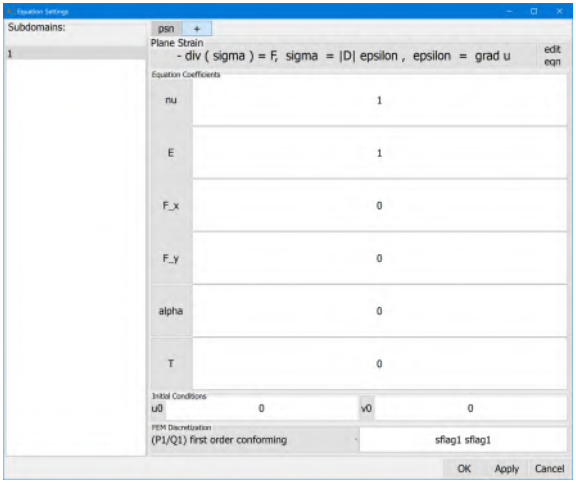
where E is the elastic or Young's modulus, and ν is the Poisson's ration of the material. The strains are related to the material displacements (u, v) as

$$\varepsilon_x = \frac{\partial u}{\partial x}, \qquad \varepsilon_y = \frac{\partial v}{\partial y}, \qquad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$$

Balance equations for the stresses finally give the resulting governing equation system as

$$\begin{cases} -\dfrac{\partial \sigma_x}{\partial x} - \dfrac{\partial \tau_{xy}}{\partial y} = F_x \\ -\dfrac{\partial \tau_{xy}}{\partial x} - \dfrac{\partial \sigma_y}{\partial y} = F_y \end{cases}$$

where F_x and F_y are volume (body) forces in the x and y-directions, respectively. In the *Equation Settings* dialog box shown below the equation coefficients, initial value for the displacements can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



Optional stress-strain temperature dependence can also be added, which takes the form

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon} - \frac{E\alpha T}{(1+\nu)(1-2\nu)}[1 \;\; 1 \;\; 0]^T$$

where α is the coefficient of thermal expansion and T either a prescribed temperature field, a dependent variable name from another physics mode that represents the temperature, or a combination such as $T - T_{ref}$.

The boundary conditions for the plane strain physics mode allows for any combination of prescribing displacements and edge loads. These boundary conditions are summarized in the table below.

Boundary Condition	Definition	Boundary Coefficient
Fixed displacements	Prescribed displacements, $u = u_0, v = v_0$	u_0, v_0
Edge loads	Prescribed edge loads, $F_x = f_{x,0}, F_y = f_{y,0}$	$f_{x,0}$, $f_{y,0}$

4.3.10 Axisymmetric Stress-Strain

The axisymmetric stress-strain physics mode models how stresses and strains in cylindrical and rotationally symmetric geometries behave. The model equations are reduced from full 3D to a two-dimensional slice. In this case the balance equations for stress-strain (valid for the half plane *r>0*, with *r=0* being the symmetry axis) can be written as

$$\begin{cases} -\dfrac{\partial \sigma_r}{\partial r} - \dfrac{\partial \tau_{rz}}{\partial z} - \dfrac{\sigma_r - \sigma_\theta}{r} = F_r \\ -\dfrac{\partial \tau_{rz}}{\partial r} - \dfrac{\partial \sigma_z}{\partial z} - \dfrac{\tau_{rz}}{r} = F_z \end{cases}$$

where F_r and F_z are volume (body) forces in the r and z-directions, respectively. The balance equations together with the constitutive relations

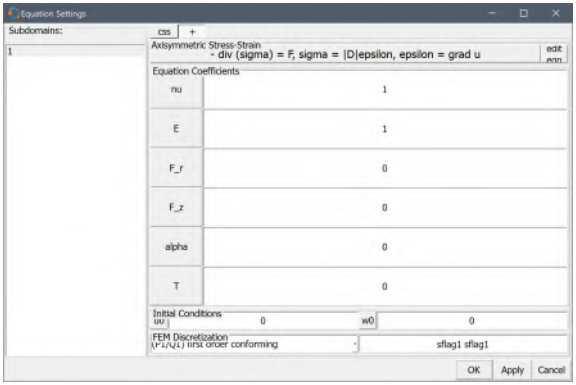
$$\begin{pmatrix} \sigma_r \\ \sigma_\theta \\ \sigma_z \\ \tau_{rz} \end{pmatrix} = \frac{E\nu}{(1+\nu)(1-2\nu)} \begin{pmatrix} \frac{1-\nu}{\nu} & 1 & 1 & 0 \\ 1 & \frac{1-\nu}{\nu} & 1 & 0 \\ 1 & 1 & \frac{1-\nu}{\nu} & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2\nu} \end{pmatrix} \begin{pmatrix} \varepsilon_r \\ \varepsilon_\theta \\ \varepsilon_z \\ \gamma_{rz} \end{pmatrix}$$

uniquely define the equations to solve. Here E is the elastic or Young's modulus, and ν is the Poisson's ratio of the material. The strains are related to the material displacements (u, w) as

$$\varepsilon_r = \frac{\partial u}{\partial r}, \qquad \varepsilon_\theta = \frac{u}{r}, \qquad \varepsilon_z = \frac{\partial w}{\partial z}, \qquad \gamma_{rz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial r}$$

In the PDE equation formulations the dependent variable *u* is replaced by *u/r* in order to avoid divisions by zero on the symmetry line (Thus custom postprocessing expressions including *u* should instead use *r∗u*).

In the *Equation Settings* dialog box shown below the equation coefficients, initial value for the displacements can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



Optional stress-strain temperature dependence can also be added, which takes the form

$$\sigma = \mathbf{D}\varepsilon - \frac{E\alpha T}{1-\nu}[1 \;\; 1 \;\; 1 \;\; 0]^T$$

where α is the coefficient of thermal expansion and T either a prescribed temperature field, a dependent variable name from another physics mode that represents the temperature, or a combination such as $T - T_{ref}$.

The boundary conditions for the plane stress physics mode allows for any combination of prescribing displacements and edge loads. These boundary conditions are summarized in the table below.

Boundary Condition	Definition	Boundary Coefficient
Fixed displacements	Prescribed displacements, $u = u_0, w = w_0$	u_0, w_0
Edge loads	Prescribed edge loads, $F_r = f_{r,0}, F_z = f_{z,0}$	$f_{r,0}$, $f_{z,0}$

4.3.11 Linear Elasticity

The linear elasticity physics mode models how structural stresses form in solid structures where the stress-strain relations can be written as

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{xz} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{pmatrix} \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{xz} \end{pmatrix}$$

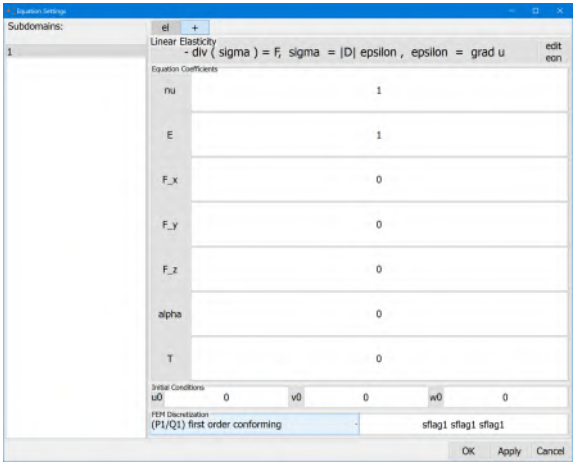
where E is the elastic or Young's modulus, and ν is the Poisson's ratio of the material. The strains are related to the material displacements (u, v, w) as

$$\varepsilon_x = \frac{\partial u}{\partial x}, \qquad \varepsilon_y = \frac{\partial v}{\partial y}, \qquad \varepsilon_z = \frac{\partial w}{\partial z}, \qquad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}, \qquad \gamma_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}, \qquad \gamma_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}$$

Balance equations for the stresses finally give the resulting governing equation system as

$$\begin{cases} -\dfrac{\partial \sigma_x}{\partial x} - \dfrac{\partial \tau_{xy}}{\partial y} - \dfrac{\partial \tau_{xz}}{\partial z} = F_x \\ -\dfrac{\partial \tau_{xy}}{\partial x} - \dfrac{\partial \sigma_y}{\partial y} - \dfrac{\partial \tau_{yz}}{\partial z} = F_y \\ -\dfrac{\partial \tau_{xz}}{\partial x} - \dfrac{\partial \tau_{yz}}{\partial y} - \dfrac{\partial \sigma_z}{\partial z} = F_z \end{cases}$$

where F_x, F_y, and F_z are volume (body) forces in the x, y, and z-directions, respectively. In the *Equation Settings* dialog box shown below the equation coefficients, initial value for the displacements can be specified. The FEM shape function space can also be selected from the drop-down combobox (1st through 5th order conforming P1/Q1 shape functions), or further specified in the corresponding edit field.



Optional stress-strain temperature dependence can also be added, which takes the form

$$\sigma = \mathbf{D}\varepsilon - \frac{E\alpha T}{1-2\nu}[1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0]^T$$

where α is the coefficient of thermal expansion and T either a prescribed temperature field, a dependent variable name from another physics mode that represents the temperature, or a combination such as $T - T_{ref}$.

The boundary conditions for the linear elasticity physics mode allows for any combination of prescribing displacements and edge loads. These boundary conditions are summarized in the table below

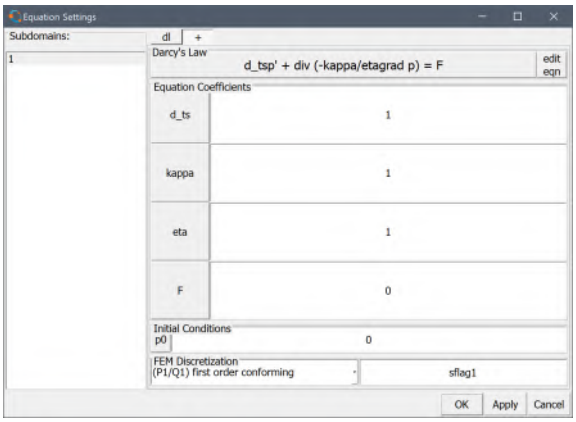
Boundary Condition	Definition	Boundary Coefficient
Fixed displacements	Prescribed displacements, $u = u_0, v = v_0, w = w_0$	u_0, v_0 , w_0
Edge loads	Prescribed edge loads, $F_x = f_{x,0}, F_y = f_{y,0}, F_z = f_{z,0}$	$f_{x,0}$, $f_{y,0}$, $f_{z,0}$

4.3.12 Darcy's Law

Darcy's law models the pressure *p* in porous media flow through the relation

$$d_{ts}\frac{\partial p}{\partial t} + \nabla \cdot (-\frac{\kappa}{\eta}\nabla p) = F$$

where κ represents permeability, η viscosity, F sources and sinks, and d_{ts} is a time scaling coefficient. In the Darcy's Law *Equation Settings* dialog box shown below the equation coefficients, initial values for the pressure can be specified. The FEM shape function spaces can also be selected from the drop-down combobox, or further specified in the corresponding edit field.



The Darcy's law physics mode allows prescription of three different boundary conditions. The prescribed pressure, flux, and insulation boundary conditions are summarized in the table below

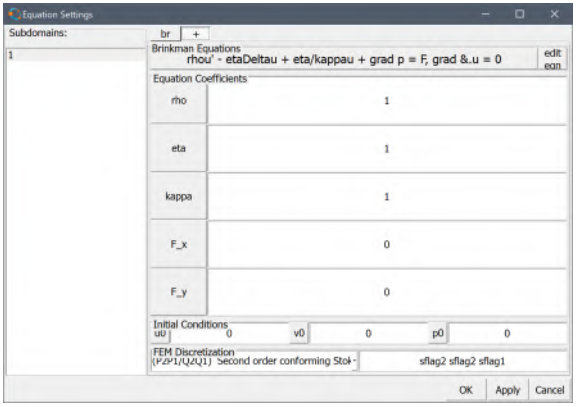
Boundary Condition	Definition	Boundary Coefficient
Pressure	Prescribed pressure, $\mathbf{p} = p_0$	p_0
Insulation/symmetry	Zero flux/flow, $-\mathbf{n} \cdot (-\frac{\kappa}{\eta}\nabla p) = 0$	
Flux	Prescribed flux/flow, $-\mathbf{n} \cdot (-\frac{\kappa}{\eta}\nabla p) = N_0$	N_0

4.3.13 Brinkman Equations

The Brinkman equations physics mode models porous media flows and can be seen as a combination of Darcy's law with the Navier-Stokes equations. The Brinkman equations are defined as follows

$$\begin{cases} \rho \dfrac{\partial \mathbf{u}}{\partial t} - \eta \Delta \mathbf{u} + \dfrac{\eta}{\kappa} \mathbf{u} + \nabla p = \mathbf{F} \\ \nabla \cdot \mathbf{u} = 0 \end{cases}$$

which is to be solved for the unknown velocity field $\mathbf{u}$ and pressure p. In these equations ρ represents the density of the fluid, η the viscosity, and κ permeability, moreover $\mathbf{F}$ represents body forces acting on the fluid. In the *Equation Settings* dialog box shown below the equation coefficients, initial values for the velocities and pressures can be specified. The FEM shape function spaces can also be selected from the drop-down combobox (P2P1/Q2Q1 and P1P-1/Q2P1 shape functions), or further specified in the corresponding edit field.



The Brinkman equations physics mode allows prescription of several different boundary conditions. Firstly, the no-slip (zero velocity) boundary condition which is appropriate for stationary walls. Moreover, a prescribed velocity condition can be prescribed to both in and outflows as well as moving walls. Prescribed pressure and neutral (zero viscous stress) conditions are both appropriate for outflows. Lastly, symmetry or slip conditions sets the flow to zero in one coordinate direction so as to prevent flow normal to the boundary. These boundary conditions are summarized in the table below

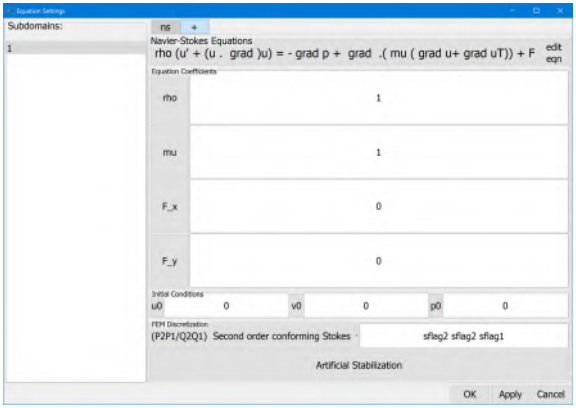
Boundary Condition	Definition	Boundary Coefficient
Wall/no-slip	Zero velocity, $\mathbf{u} = 0$	
Inlet/velocity	Prescribed velocity, $\mathbf{u} = (u_0, v_0, w_0)$	u_0, v_0, w_0
Neutral outflow/stress boundary	Zero stress, $-\mathbf{n} \cdot (-p\mathbf{I} + \eta \nabla \mathbf{u}) = 0$	
Outflow/pressure	Prescribed pressure, $p = p_0$	p_0
Symmetry/slip, x_i-direction	Zero velocity in x_i-direction, $u_i = 0$	

4.3.14 Navier-Stokes Equations (Incompressible Fluid Flow)

The Navier-Stokes equations physics mode models flows of incompressible fluid flows and which is described by

$$\begin{cases} \rho \left(\dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) - \nabla \cdot (\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)) + p = \mathbf{F} \\ \nabla \cdot \mathbf{u} = 0 \end{cases}$$

which is to be solved for the unknown velocity field $\mathbf{u}$ and pressure p. In these equations ρ represents the density of the fluid and μ the dynamic viscosity, moreover $\mathbf{F}$ represents body forces acting on the fluid. In the *Equation Settings* dialog box shown below the equation coefficients, initial values for the velocities and pressures can be specified. The FEM shape function spaces can also be selected from the drop-down combobox (P2P1/Q2Q1 and P1P-1/Q2P1 shape functions), or further specified in the corresponding edit field.



For flow problems with dominating convective effects it is advisable to use artificial stabilization. Pressing the lower **Artificial Stabilization** opens the corresponding dialog box which allows adding both isotropic artificial diffusion and anisotropic streamline diffusion. Turning coefficients are also provided to control the strength of the introduced artificial diffusion.

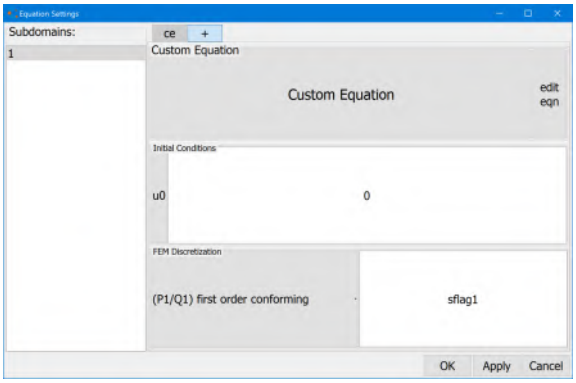
The Navier-Stokes equations physics mode allows prescription of several different boundary conditions. Firstly, the no-slip (zero velocity) boundary condition which is appropriate for stationary walls. Moreover, a prescribed velocity condition can be prescribed to both in and outflows as well as moving walls. Prescribed pressure and neutral (zero viscous stress) conditions are both appropriate for outflows. Lastly, symmetry or slip conditions sets the flow to zero in one coordinate direction so as to prevent flow normal to the boundary. These boundary conditions are summarized in the table below

Boundary Condition	Definition	Boundary Coefficient
Wall/no-slip	Zero velocity, $\mathbf{u} = 0$	
Inlet/velocity	Prescribed velocity, $\mathbf{u} = (u_0, v_0, w_0)$	u_0, v_0, w_0
Neutral outflow/stress boundary	Zero stress, $-\mathbf{n} \cdot (-p\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)) = 0$	

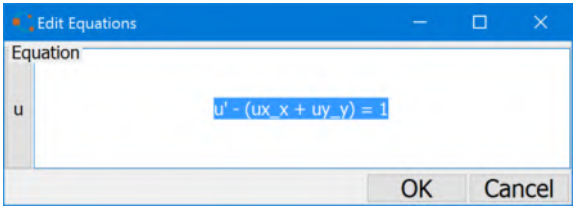
Outflow/pressure	Prescribed pressure, $p = p_0$	p_0
Symmetry/slip, x_i-direction	Zero velocity in x_i-direction, $u_i = 0$	

4.3.15 Custom Equation

User defined equations can be prescribed by using the custom equation physics mode.



The equation specification can be accessed by either pressing the **edit** button next to the equation description, or the equation description itself. A dialog box will appear showing one edit field for equation and corresponding dependent variable.



Note that the other physics mode equations are defined similarly and can also be edited by pressing the edit button in the corresponding tabs of the equation settings dialog box.

The syntax for equation specifications tries to as close as possible look like how one would write a partial differential equation with pen and paper. If for example the dependent variable is **u** like in the example above, then **u'** corresponds to the time derivative, **ux** the derivative in the x-direction, **ux_x** a second order derivative in the x-direction (to which partial integration will be applied according to the standard finite element derivation of the weak formulation), and **u_t** is the variable u multiplied with the fem test function, thus it will be assembled to the iteration matrix instead of the right hand side. The following table describes the syntax and legal operators

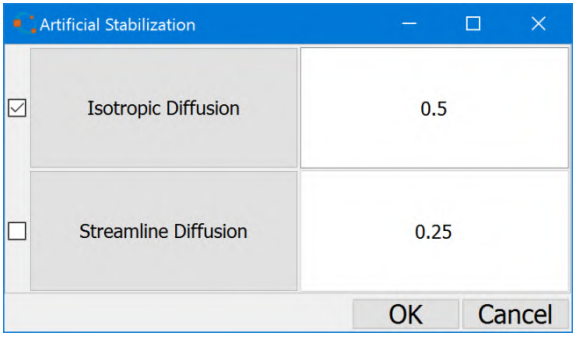
Syntax	Description	Formula
u	Dependent variable name	u
x	Space dimension name	x
ux	Derivative in x-direction, rhs	$\frac{\partial u}{\partial x}$
u_t	Dep. var multiplied with test function	$u \cdot v$
u_x	Derivative of test function	$u \cdot \frac{\partial v}{\partial x}$
ux_t	Derivative in x-direction	$\frac{\partial u}{\partial x} \cdot v$
ux_x	2nd derivative in x-direction	$\frac{\partial u}{\partial x} \cdot \frac{\partial v}{\partial x}$
uxx_x	3rd derivative in x-direction	$\frac{\partial^2 u}{\partial x^2} \cdot \frac{\partial v}{\partial x}$
ux_xx	3rd derivative in x-direction	$\frac{\partial u}{\partial x} \cdot \frac{\partial^2 v}{\partial x^2}$
uxx_xx	4th derivative in x-direction	$\frac{\partial^2 u}{\partial x^2} \cdot \frac{\partial^2 v}{\partial x^2}$
+	Addition	
-	Subtraction	
*	Multiplication	
/	Division	
sqrt()	Square root	
^	Power	
()	Delimit by enclosing in parentheses	

where *v* is a test function. The equation syntax parser accepts numeric constants and coefficients defined in the **fea.coef** field. Higher dimensions work analogously and the space dimension **x** can be substituted with the others arbitrarily. For a more complicated example look at Custom Equation - Black-Scholes model equation in the tutorials section.

4.4 Artificial Stabilization

For problems with dominating convective effects such as can be found in convection and diffusion, heat transfer problems with convection, and fluid flow problems one can employ artificial stabilization if the grid size is too coarse to allow convergence.

The **Artificial Stabilization** dialog box allows control of both isotropic artificial diffusion and anisotropic streamline diffusion options. Turning coefficients are also provided to control the strength of the introduced artificial diffusion.



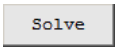
Isotropic artificial diffusion adds diffusion of magnitude **delta**∗**h_grid**∗|**u**| in all directions, where **delta** is the tuning coefficient, **h_grid** the local mean diameter of a grid cell, and |**u**| the magnitude of the convective velocity. Streamline diffusion modifies the finite element test function space and only adds a stabilization coefficient of the form **delta**∗**h_grid**/|**u**| in the direction of the flow so as to minimize changes to the original problem.

5 Solvers

This chapter describes the solver options available with FEATool Multiphysics. Solvers are used to compute solutions to the involved discretized equations.

The default option in FEATool is to use the best linear solvers available in Matlab and Octave, which currently is the Umfpack and SuiteSparse direct linear solvers. These solvers are very robust for a wide range of problems, but are not as memory efficient or fast as specialized and dedicated solvers can be. Due to this FEATool also features GUI and CLI interfaces to specialized external solvers, such as the computational fluid dynamics (CFD) solver OpenFOAM, and the general FEA and PDE solver FEniCS. These interfaces allows one to set up and define multiphysics problems in Matlab and Octave with the FEATool GUI and CLI interfaces, export and solve them with the external solvers, and automatically import the solutions for further (post-)processing. The OpenFOAM and FEniCS external solvers are discussed in the following sections.

5.1 Solve Mode

After geometry, grid, and equation and boundary conditions have been defined one can change to solve mode by pressing the 

mode button.

The **Solve** menu similarly allows for switching to **Solve Mode** and also opening the **Solver Settings...** dialog box. The *Solve* menu also features an option to **Get Initial Solution** which computes the expressions defined in the *Initial Condition* fields of the equation settings dialog

box, and returns this as the solution. Moreover, the default solver call command can be edited and changed by using the **Solver Call Command...** option, allowing for custom and external solvers to be used.

In solve mode the available toolbar buttons and actions are the following.

- Calls the default solver as defined by the *solver settings* with initial conditions prescribed in the equation settings. After the problem has been solved FEATool switches to *postprocessing* mode and plots the solution according to the chosen plot types.

- Re-solves the problem using the last solution as initial value. Note that for this to work the geometry, grid, physics modes, and finite element shape functions must all remain the same.

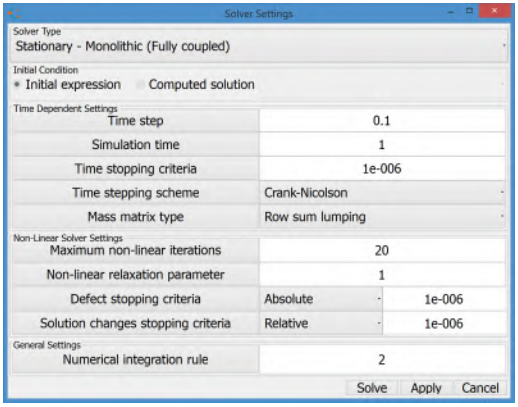
- Solves fluid flow problems with the `OpenFOAM` external CFD solver.

- Solves general finite element problems with the `FEniCS` external FEA solver.

- Opens the *Solver Settings* dialog box.

5.2 Solver Settings Dialog Box

The **Solver Settings** dialog box shown below and can be opened by pressing the corresponding toolbar button. The solver settings allow advanced control over the default built-in FEATool solver.



Firstly, the **Solver Type** option allows for either stationary monolithic solver solvestat, or the time dependent instationary solver solvetime to be chosen. Monolithic means that all dependent variables are discretized and solved together in a large coupled system, rather than iteratively solving a segregated set of smaller decoupled systems.

Initial Condition controls whether the initial values should be computed from the equation expressions in the initial value fields, or taken from an old solution in which case the solution can be chosen from the corresponding list box. Note that for this to work the geometry, grid, physics modes, and finite element shape functions must all remain the same.

The **Time Dependent Settings** controls time-stepping parameters when using the time-dependent solver. The **Time step** selects a fixed time step size for single step schemes, or the average time step size for multi-step schemes. Time dependent simulations will terminate either when the accumulated time has reached the specified **Simulation time**, or the normed changes in the dependent solution variables are smaller than the **Time stopping criteria** indicating a stationary state. The **Time stepping scheme** allows choosing between the single-step first order Backward Euler and second order Crank-Nicolson schemes, as well as the second order (multi-step) Fractional-step-theta scheme. Lastly, one can choose between using standard row-sum lumped and full **Mass matrix** representations.

The non-linear solver is used then dependent variables are detected in the equation coefficients, which is the typical for multiphysics problems. Per default standard fixed point iterations are used while Newton iterations can be used if an analytical Jacobian is supplied by the user (see the Navier-Stokes fluid flow models for examples of this).The **Non-Linear Solver Settings** allows control over the **Maximum number of non-linear iterations**, and the **Non-linear relaxation parameter** which is the fraction of the new to old solution to weigh in the next non-linear

iteration. The non-linear solver will terminate and stop when either of the **Defect stopping criteria** or **Solution changes stopping criteria** are met which can be chosen to be measured both absolute or relatively. Stopping criteria for the non-linear solver

In the **General Settings** frame one there is a setting to control the numerical integration rule and order used in the matrix assembly operations (it is necessary to increase this for higher order >2 shape functions).

5.3 OpenFOAM CFD Solver

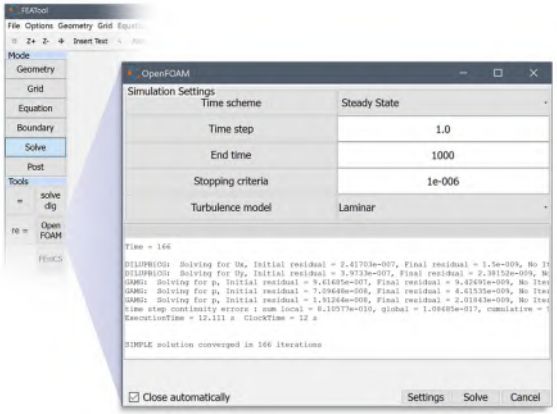
OpenFOAM, is very popular, flexible, and efficient finite volume FVM based computational fluid dynamics (CFD) solver for both incompressible and compressible laminar and turbulent flow regimes. The stable and tested OpenFOAM code base has been used and validated in many academic and commercial CFD projects.

The OpenFOAM-FEATool CFD solver integration makes it easy to perform high performance CFD simulations directly in Matlab and Octave, typically yielding a magnitude or more speedup compared to the built-in solvers. Additionally, iterative and multigrid solvers, such as included in OpenFOAM, uses much less memory than direct solvers allowing for much larger simulations.

5.3.1 OpenFOAM Usage

To use OpenFOAM to solve Navier-Stokes fluid flow problems, press the corresponding *Solve Mode* toolbar button labeled **OpenFOAM**, instead of the default solve button $**=**$. This will open the OpenFOAM solver settings and control dialog box.

The *OpenFOAM* solver dialog box allows one to automatically use *OpenFOAM* to solve CFD problems. The **Solve** button automatically export, solves and re-imports the computed solution into FEATool. If the **Close automatically** checkbox is marked, the OpenFOAM dialog box will be automatically closed and FEATool switch to Postprocessing mode after the solution has been computed. **Settings** option allows for manually editing the OpenFOAM control dictionary and case files and overriding the default settings.



The following OpenFOAM solver settings can be controlled though the dialog box

- **Time scheme** - Selects between *Steady State*, *Euler*, *Crank-Nicolson*, *Backward*, and *Local Euler* time discretization schemes.
- **Time step** - Specifies the time step size.
- **End time** - Sets the simulation maximum end time.
- **Stopping criteria** - Specifies the stopping criteria for steady state simulations (corresponding to the *residualControl* parameter).
- **Turbulence model** - Selects between the pre-defined turbulence models **Laminar** and **k-epsilon**.

5.3.2 Command Line (CLI) Usage

The openfoam function can also be used on the command line as follows:

FEA = OPENFOAM(FEA, VARARGIN) Export, solves, or imports the solved problem described in the FEATool FEA problem struct using the OpenFOAM CFD solver. Accepts the following property/value pairs, moreover also accepts the openfoam_data property/value pairs to set the OpenFOAM dicts during export.

```
Input          Value/{Default}        Description
-------------------------------------------------------------------------
modes          check, export,         Command mode(s) to call
               solve, import
data           default                Default OpenFOAM data and parameter dict
application    string {simpleFoam}    OpenFOAM application binary to run
ddtScheme      string {steadyState}   Time stepping scheme
tolres         scalar/vector {1e-5}   Stopping criteria for residuals (Simple)
startTime      scalar {0.0}           Simulation start time
endTime        scalar {100}           Simulation end time
deltaT         scalar {1.0}           Time step size
maxDeltaT      scalar {0.1}           Maximum time step size
maxCo          scalar {0.5}           Maximum Courant number
writeInterval  scalar {100}           Solution output write interval
purgeWrite     scalar {0}             Specified number of output solutions
transportModel string {Newtonian}     Transport model
simulationType string {laminar}       Simulation type laminar/RAS/LES
RASModel       string {kEpsilon}      RAS turbulence model
LESModel       string {Smagorinsky}   LES turbulence model
nu             scalar {1.0}           Kinematic viscosity (constant)
interp         true                   Interpolate solution to grid points
casedir        default                OpenFOAM case directory
foamdir        default                OpenFOAM installation directory
logfname       featool-openfoam       OpenFOAM log/output filename
logfid         1                      Log file/message output file handle
tolres         1e-5                   Stopping criteria for initial residuals
clear          boolean {true}         Clear output and log files
```

5.3.3 Installation

Note that the OpenFOAM solver binaries are not included with the FEATool distribution and must be compiled and installed separately.

For Microsoft Windows systems it is generally recommended to install and use the pre-compiled blueCFD-core (2017) binaries from blueCAPE http://bluecfd.com/Core. For Ubuntu

`Linux OpenFOAM` can be installed with the *apt-get* package manager

```
sudo add-apt-repository http://dl.openfoam.org/ubuntu
sudo sh -c "wget -O - http://dl.openfoam.org/gpg.key | apt-key add -"
sudo apt-get update
sudo apt-get -y install openfoam5
```

For other systems, please follow the installation recommendations from the official `OpenFOAM website http://openfoam.org`.

The FEATool openfoam function will attempt to call the OpenFOAM solver binaries *simpleFoam* and *pimpleFoam* through the use of the *openfoam.sh* and *openfoam.bat* shell script which may be modified to suit the users need (which can be found in the *openfoam* folder).

5.3.4 Limitations

The following limitations currently apply to the OpenFOAM solver.

- Single Navier-Stokes physics mode
- Single geometry object with one subdomain
- Constant density and viscosity
- Constant initial conditions
- Constant velocity/pressure boundary conditions

5.3.5 Advanced Usage

The OpenFOAM solvers are capable of performing large scale simulations, and although technically possible to use FEATool to do this, for memory and stability reasons it is not advised to do this from within Matlab or Octave.

For large scale simulations then it is instead recommended to first export a FEATool model with the openfoam export command. This will generate corresponding OpenFOAM case files. Then one can manually launch the OpenFOAM solver the system command line, and possibly manually import the solution when the solution has finished (consult the OpenFOAM documentation for a full explanation how to use OpenFOAM as a stand alone solver).

5.3.6 Tutorial

An example of setting up and solving a flow over a backwards facing step problem using the OpenFOAM external solver is given in the tutorial section Fluid Dynamics - Flow Over a Backwards Facing Step.

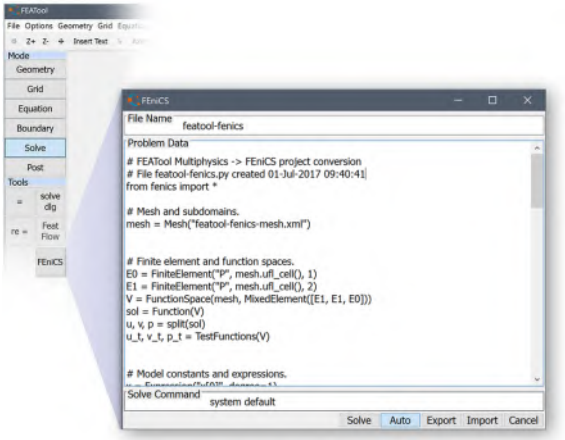
5.4 FEniCS General FEA Solver

FEniCS is a flexible and comprehensive finite element FEM and partial differential equation PDE modeling and simulation toolkit with Python and C++ interfaces along with many integrated solvers. As both FEATool and FEniCS discretize equations employing a weak finite element formulation it is quite straightforward to translate FEATool syntax and convert it to FEniCS *Python* scripts. Similar to what has been done with the OpenFOAM CFD solver the FEATool-FEniCS integration allows for easy conversion, exporting, solving, and importing FEATool Multiphysics models to FEniCS directly from the GUI, as well as the Matlab and Octave command line interfaces.

FEniCS is aimed at supporting and solving general systems of PDEs, such as found in engineering and coupled multiphysics problems. As both FEATool and FEniCS discretize the equations in the same way the solutions they produce should be virtually identical. One of the many advantages of using FEniCS then, in addition to now supporting the *Python* scripting language as well as Matlab and Octave, is that FEniCS solvers have support for both distributed (MPI) and shared memory (OpenMP) parallel execution allowing for larger models to be solved faster (FEniCS has been tested on problem sizes up to 10^8 degrees of freedom run on 512 CPUs in parallel, where Matlab and Octave in contrast is limited to the serial sparse linear solvers, the Umfpack direct solver and a selection of iterative ones.

5.4.1 GUI Usage

A button labeled **FEniCS** is present in the *Solve Mode* toolbar of the FEATool GUI. Pressing this button will open a dialog box where the current model equations and parameters has been translated to a FEniCS *Python* script. The script can there be inspected and edited, as well as changing the output file name and system shell command to run it (which is *bash* by default).



The FEniCS solver dialog box also features **Solve**, **Auto**, **Export**, and **Import** buttons. If the **Auto** toggle button is engaged the **Solve** button will automatically try to **Export**, **Solve**, and **Import** the computed solution. With the **Auto** toggle disengaged the **Solve** button only execute the FEniCS *Python* script in a bash shell if present (not export or import). The **Export** option saves the current grid in Dolfin XML format and exports the FEniCS simulation script and **Import** will import an existing solution if it matches with the FEATool problem definition.

Additional options include the base *File Name* and overriding the default bash *Solve Command*.

5.4.2 Command Line Usage

The installed fenics Matlab function can also be used on the command line (CLI) to manually perform the export, solve, and import actions. In the following is an example to set up a simple heat transfer model on a unit circle with a unit heat source term, and *T=0* fixed temperature on all the boundaries. In the FEATool Matlab m-script language the model can look like the following

```
fea.sdim = {'x' 'y'};
fea.grid = quad2tri(circgrid());

fea = addphys(fea, @heattransfer);
fea.phys.ht.eqn.coef{6,end} = {1};   % Heat source term.
fea.phys.ht.bdr.sel(:) = 1;          % Zero temperature boundary conditions.

fea = parsephys(fea);                % Parse physics mode and fea problem stru
fea = parseprob(fea);

fea.sol.u = solvestat(fea);

postplot(fea, 'surfexpr', 'T')
```

By running the command **fenics(fea, 'mode', 'export')** a grid file *featool-fenics-mesh.xml* and FEniCS simulation script file *featool-fenics.py* will automatically be generated in the current directory

```
from fenics import *

# Mesh and subdomains.
mesh = Mesh("featool_fenics_mesh.xml")

# Finite element and function spaces.
E0 = FiniteElement("P", mesh.ufl_cell(), 1)
V = FunctionSpace(mesh, E0)
T = Function(V)
T_t = TestFunction(V)

# Model constants and expressions.
rho = Constant(1)
```

```
cp = Constant(1)
k = Constant(1)
u = Constant(0)
v = Constant(0)
q = Constant(1)

# Bilinear forms.
a = ( (rho*cp*u)*T*T_t.dx(0) + (k)*T.dx(0)*T_t.dx(0) + \
      (rho*cp*v)*T*T_t.dx(1) + (k)*T.dx(1)*T_t.dx(1) )*dx

# Linear forms.
f = q*T_t*dx

# Boundary conditions.
dbc0 = DirichletBC(V, Constant(0), 0)
dbc1 = DirichletBC(V, Constant(0), 1)
dbc2 = DirichletBC(V, Constant(0), 2)
dbc3 = DirichletBC(V, Constant(0), 3)
dbc = [dbc0, dbc1, dbc2, dbc3]

# Initial conditions.
assign(T, interpolate( Constant(0), V))

# Solve.
solve( a - f == 0, T, dbc )

# Output.
import numpy as np
T_h = T.compute_vertex_values(mesh)
np.savetxt("featool_fenics_sol.txt", np.column_stack((T_h)))

# Postprocessing.
plot(T, title = "T")
interactive()
```

The generated FEniCS *Python* FEM script is longer and somewhat more verbose since all the FEATool physics mode defaults must be explicitly expressed.

Calling **fenics(fea, 'mode', 'solve')** will attempt to solve the problem if *Python* and FEniCS is installed and set up correctly. Alternatively, the script can be run by itself by a valid FEniCS installation.

The solution process will generate the file *featool-fenics-sol.txt* containing the nodal solution(s) which can be imported back into FEATool with **fea = fenics(fea, 'mode', 'import')** after which it can be postprocessed and visualized with the usual FEATool and Matlab functions.

5.4.3 Notes

The FEniCS-FEATool integration and problem file export should work for general multiphysics problems using both the GUI and command line. However, a subset of FEATool functionality is currently not supported (the known ones are listed here)

- Non-linear Dirichlet fixed value boundary conditions are not supported. The exported FEniCS C++ boundary condition expressions do not currently allow for dependent variables / solution unknowns.
- As FEniCS does not feature a built-in time dependent solver. Solvers for instationary problems must currently be implemented manually.
- Switch and logical expressions (for example *x>1 & y<=0*) must be manually rewritten to conform with the C++ expression syntax.
- Point source terms are not supported by the FEniCS non-linear solution form (which is used by default since it also handles linear problems).
- Although FEATool and FEniCS uses the same finite element FEM basis functions the internal ordering is different, thus the FEniCS solution will be exported in the grid points (*P1* space). Thus even if higher-order FEM spaces are used solution accuracy will be lost during FEATool import. This issue might not be that important for visualization but when calculating quantities, boundary and subdomain integrals, and expression evaluations it would currently be more accurate to perform that in FEniCS and *Python*.
- FEniCS does not currently support quadrilateral or hexahedral grid cells and must be converted to triangles and tetrahedra, respectively. Grid conversion of these types of grids can be performed directly in the GUI or with the quad2tri and hex2tet commands. Typically this will not be an issue since FEATool and the automatic mesh generator gridgen by default creates simplex grids (line segments, triangles, and tetrahedra).

- In addition to supporting FEniCS, the exported *Python* simulation scripts should also be compatible with the `Firedrake` project solver which also uses the FEniCS Unified Form Language (UFL) for problem definitions.

5.4.4 Installation

Note that the FEATool distribution does not include a *Python* interpreter or FEniCS itself which must be installed separately. The FEniCS homepage provides instructions `how to install FEniCS on *nix systems` and pre-configured Docker Linux images.

For systems running *Windows 10* FEniCS can be installed with the `Ubuntu Bash Windows Subsystem for Linux` by simply opening a Windows Bash shell and running the FEniCS on Ubuntu commands (which automatically also installs required dependencies such as the *Python* programming language interpreter)

```
sudo add-apt-repository ppa:fenics-packages/fenics
sudo apt-get update
sudo apt-get install --no-install-recommends fenics
sudo apt-get dist-upgrade
```

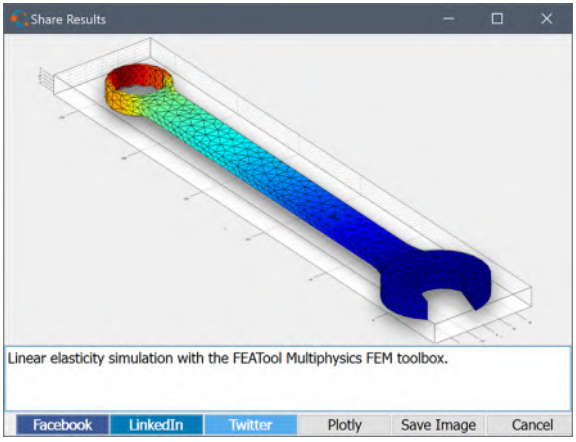
To allow FEniCS and plotting and visualization with *Python* on Windows it is also necessary to install an X window server such as `Xming` (note that plotting and output of solution variables is disabled in the FEATool-FEniCS script export by default).

6 Postprocessing

The last step in the simulation process is postprocessing and visualization. FEATool automatically switches to postprocessing mode after a solution has been computed. Alternatively, one can press the Post mode button to manually switch to postprocessing mode.

The toolbar button and corresponding menu option opens the **Postprocessing Settings** dialog box which controls various postprocessing options explained in the section below.

Images or snapshots of the current plot can be saved in jpeg or png formats by using the Share toolbar button. In the corresponding dialog box sharing results to Facebook, LinkedIn, Twitter and, Plotly is also supported.

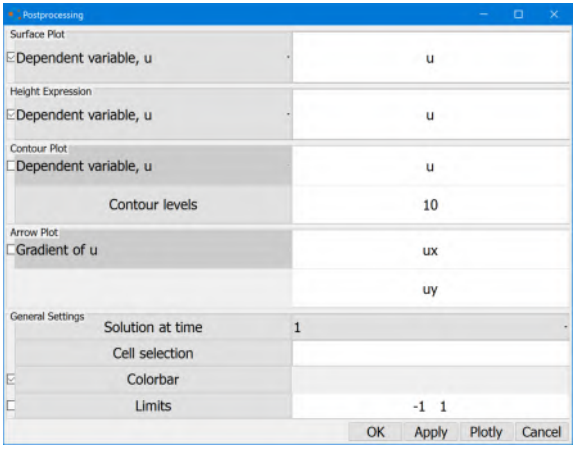


Alternatively, the **Plotly** button renders the current visualization with the `Plotly` javascript library and opens a corresponding html web page in the default web browser. The Plotly plot can be actively inspected, manipulated, exported, and shared via the plotly cloud.

The **Post** menu allows for switching to **Postprocessing Mode**, opening the **Postprocessing Settings...** dialog box, and advanced postprocessing options such as subdomain and boundary integration, and export and interfacing with external tools as explained in the section below.

6.1 Postprocessing Settings

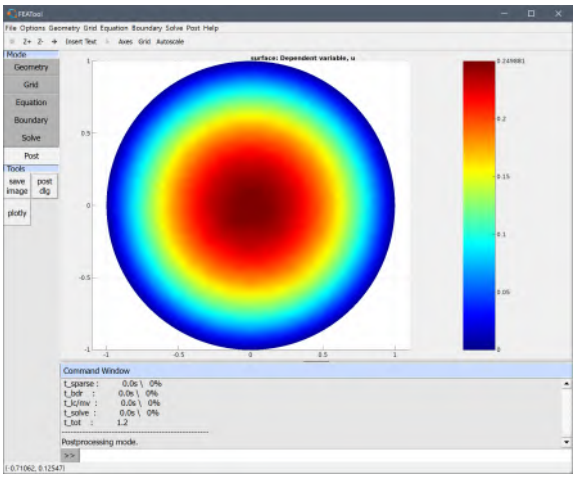
The visualization options in **Postprocessing Settings** dialog box shown in the following figure include choosing pre-defined or entering user defined custom expressions for surface, height, contour, and arrow plots. Moreover, there are options for choosing the solution to display (for time dependent solutions), selecting or excluding which cells to display, setting the min and max plot scale limits, and switching the colorbar on and off. The options are explained in more detail below.



Note that as is usual in FEATool postprocessing expressions accept general expressions involving dependent variables, space dimensions, constants, and mathematical operators. Expressions such as *v+ux+sin(2∗pi∗y)* where *u* and *v* are dependent variables (so that *ux* is the derivative in the x-direction), are perfectly valid.

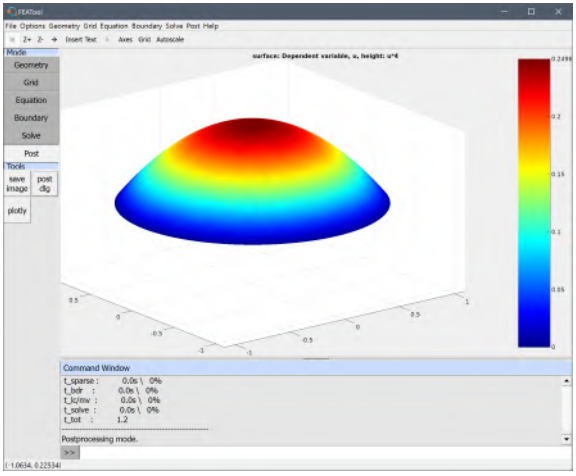
6.1.1 Surface Plot

Surface plots allows visualization of both predefined postprocessing expressions and user defined ones on the domain surface. To use the surface plot option check the corresponding check box and enter the expression to visualize.



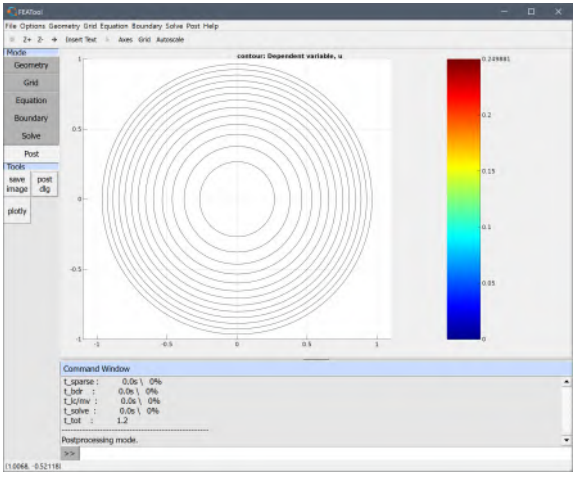
6.1.2 Height Expression

In two dimensions the height expression option is available which scales a surface plot in the z-direction with the computed value of the height expression. To use the height expression option check the corresponding check box.



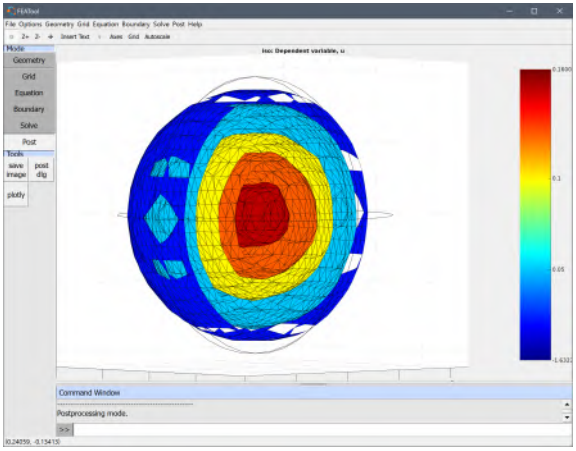
6.1.3 Contour Plot

Two dimensional problems also allow for contour or isoline plots. To use the contour plot option check the corresponding check box and select or enter an expression for the contour values. The contour values can be specified in the *Contour levels* edit field, either as an integer for the number of contours, or a space separated vector to specify fixed values for the contour levels.



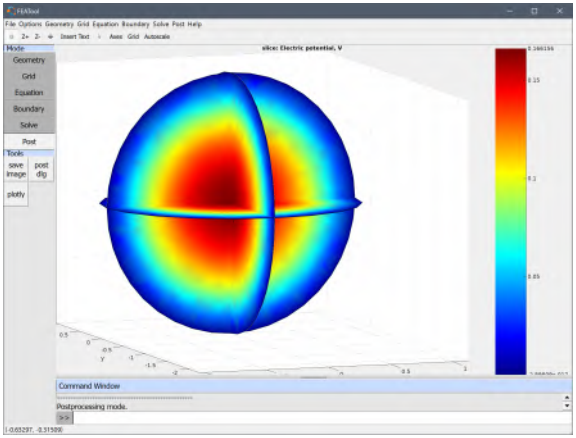
6.1.4 Iso Plot

Similarly to contour plots, three dimensional problems allow for isosurface plots. To use the isosurface plot option check the corresponding check box and select or enter an expression for the isosurface values. The iso plot values can be specified in the *Iso levels* edit field, either as an integer for the number of isosurfaces, or a space separated vector to specify fixed values for the iso levels.



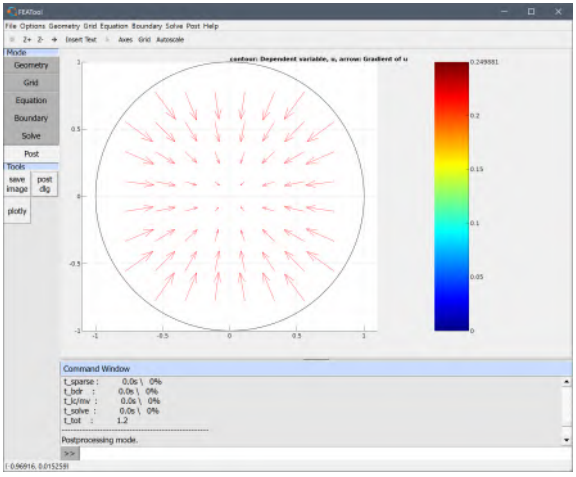
6.1.5 Slice Plot

Three dimensional problems also allow for slice plots which are surface plots perpendicular to the coordinate axes. To use the slice plot option check the corresponding check box and select or enter a slice expression. The *Slice positions* can be prescribed as a single integer signifying the number of equally spaced slices, or a space separated list indicating the exact position of each slice for the chosen axis.



6.1.6 Arrow Plot

Arrow plots are available in two and three dimensions. To use arrow plots check the corresponding check box and select or enter expressions for the vector field defining the direction of the arrows.



6.1.7 General Settings

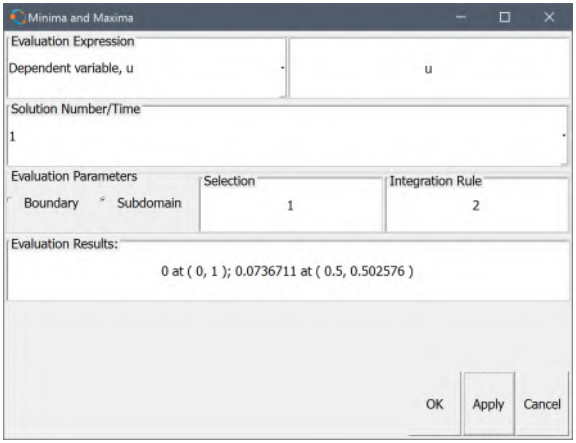
General settings allow for selecting **Solution time** if several solutions are available as with time-dependent problems. Moreover, with **Cell selection** one can limit the postprocessing to a certain subset of cells by entering a logical expression. For example, the cell selection expression *x>0* would only plot cells on the positive x-axis. The **Colorbar** check box toggles the colorbar on and off, and the **Limits** vector indicates and prescribes the minimum and maximum values of the colorbar range.

6.2 Advanced Postprocessing

The **Post** menu allows for the following advanced postprocessing options.

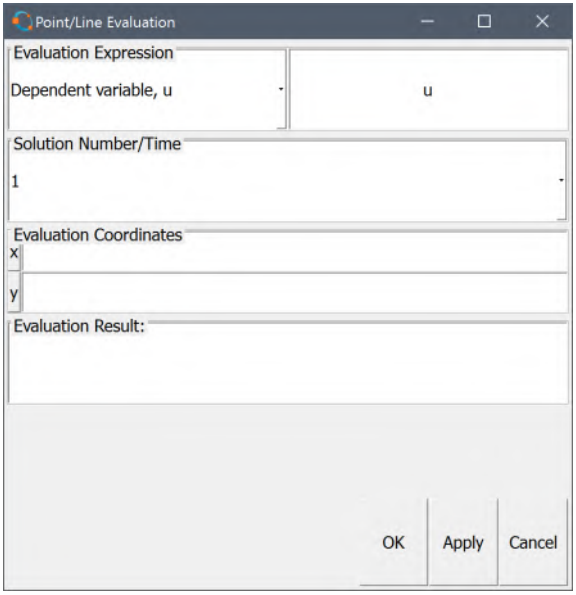
6.2.1 Min/Max Evaluation

Evaluates the minimum and maximum values for expressions on subdomains or boundaries. The **Evaluation Expression** can either be selected from a list of predefined postprocessing expressions, or entered as a user defined custom expression in the corresponding edit field. The **Solution Number/Time** can also be selected if more than one is present. In the **Selection** edit field a space separated list of **Subdomains** or **Boundaries** is specified for which the minima and maximia should be evaluated. The numerical **Integration Rule** prescribes how many evaluation points should be used for each grid cell in the evaluation process. The result will be printed in the **Evaluation Result** field as well as the command log in the main Gui.



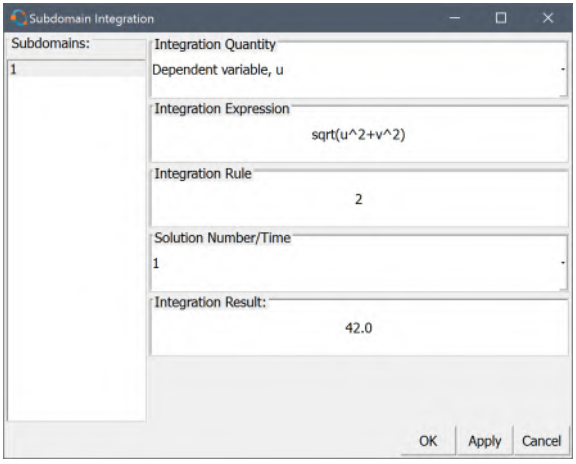
6.2.2 Point/Line Evaluation

Allows both pre and user-defined expressions to be evaluated on single points or lines. Coordinates for the evaluation points can be entered as space separated vectors for each space dimension in the corresponding edit fields. The **Solution Number/Time** to evaluate can be selected in the drop-down listbox. The result will be printed in the **Evaluation Result** field as well as the command log in the main Gui.



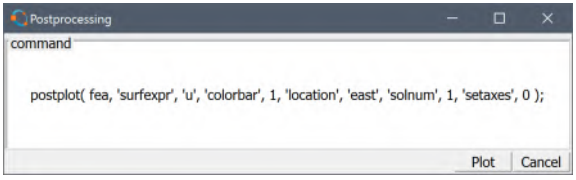
6.2.3 Boundary/Subdomain Integration

Integration of postprocessing expressions over subdomains **Subdomain Integration...** (calling intsubd) and boundaries **Boundary Integration...** (calling intbdr) can be performed by selecting the corresponding menu options. The integration dialog box allows for selecting predefined postprocessing or entering user defined expressions in the **Integration Expression** edit field. Numerical **Integration Rule**, **Solution Number/Time**, and **Subdomain/Boundary:** to integrate over can also be set. The result will be printed in the **Integration Result** field as well as the command log in the main Gui.



6.2.4 Postprocessing Call

The default postplot postprocessing function call command can be edited and changed with the **Postprocessing Call...** menu option.



6.2.5 Export Results

The solution and postprocessing data can be exported to be used with the `General Mesh Viewer` (GMV) through the **Export Results...** > **GMV Format...** menu option.

7 Tutorials

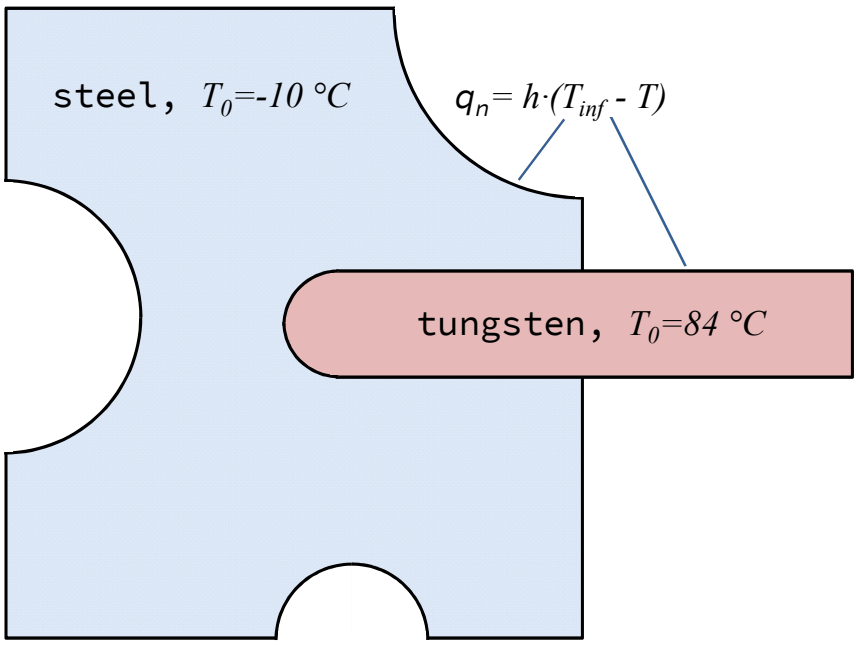
A selection of tutorial models and examples are presented in this section.

7.1 Heat Transfer - Shrink Fitting of an Assembly

FEATool supports modeling heat transfer through both conduction, that is heat transported by a diffusion process, and also convection, which is heat transported through a fluid through convection by a velocity field. The heat transfer physics mode supports both these processes, and defines the following equation

$$\rho C_p \frac{\partial T}{\partial t} + \nabla \cdot (-k \nabla c) = Q - \rho C_p \mathbf{u} \cdot \nabla T$$

where ρ is the density, C_p the heat capacity, k is the thermal conductivity, Q heat source term, and $\mathbf{u}$ a vector valued convective velocity field.



This example models heat conduction in the form of transient cooling for shrink fitting of a two part assembly. A tungsten rod heated to *84 C* is inserted into a -10 C chilled steel frame part. The time when the maximum temperature has cooled to 70 C should be determined. The assembly is cooled due to convection through a surrounding medium kept at T_{inf} = *17 C* and a heat transfer coefficient of h = *750 W/m^{2}K*. The surrounding cooling medium is not modeled directly, thus the convective term is omitted, but the effects are incorporated into the model by the use of natural convection boundary conditions [1].

7.1.1 Heat Transfer - Shrink Fitting of an Assembly using the GUI

This section describes how to set up and solve the thermal shrink fitting example with the FEATool graphical user interface (GUI).

1. Start Octave or Matlab, and if you have not run the installation script (which automatically adds the FEATool directory paths at startup) then change your working directory to where your FEATool installation is, for example

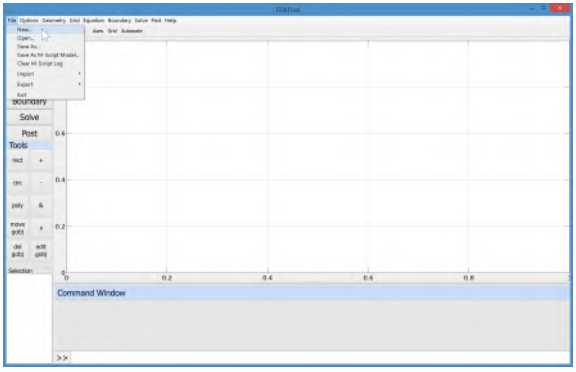
   ```
   cd C:\featool
   ```

2. In the command window type

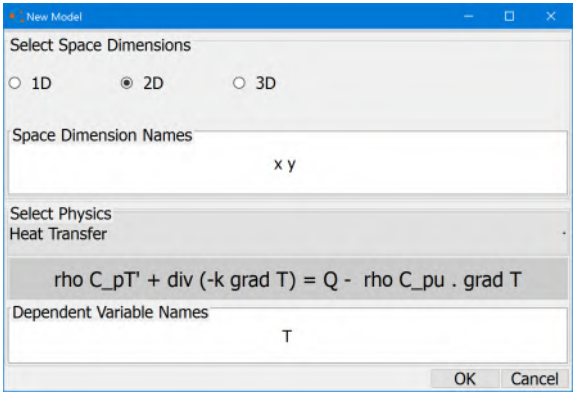
   ```
   featool
   ```

   to start the graphical user interface (GUI).

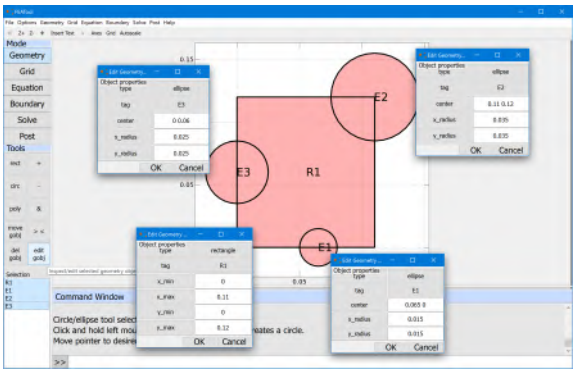
3. Click on the **New Model** button in the upper horizontal toolbar to clear all data and start defining a new model.



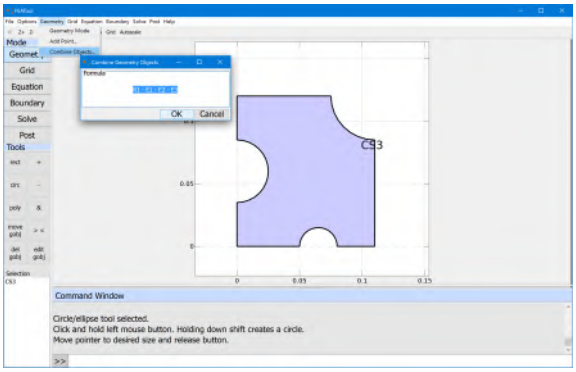
4. In the opened *New Model* dialog box, click on the **2D** radio button in the *Select Space Dimensions* frame, and select **Heat Transfer** from the *Select Physics* drop-down list. Leave the space dimension and dependent variable names to their default values. Finish and close the dialog box by clicking on the **OK** button.



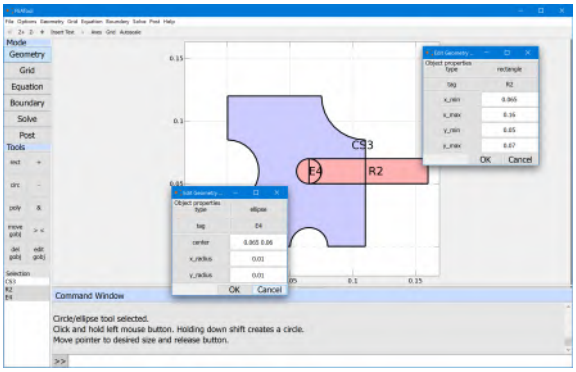
5. To create the first steel part, use the **Create square/rectangle** , **Create circle/ellipse** tools to create one rectangle and three circles. The **Inspect/edit selected geometry object** toolbar button allows access to edit geometry object dimensions and properties. Give the rectangle dimensions so that it spans it spans between 0 and 0.11 in the x-direction, and 0 and 0.12 in the y-direction. Similarly, set the circles radii to 0.015, 0.035, 0.025, and centers to (0.065, 0), (0.11, 0.12), and (0, 0.06).



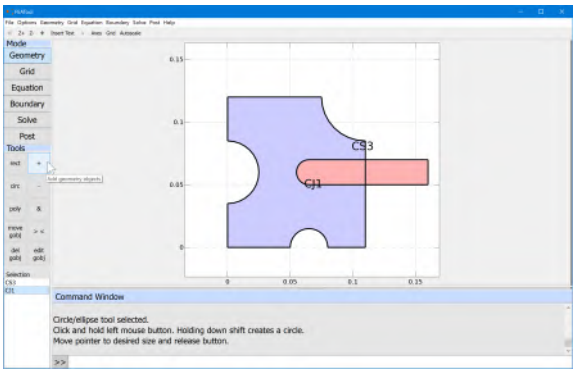
6. Generate a composite object for the part by using **Combine Objects...** from the **Geometry** menu. By using the formula **R1-E1-E2-E3** all circles will be subtracted from the rectangle.



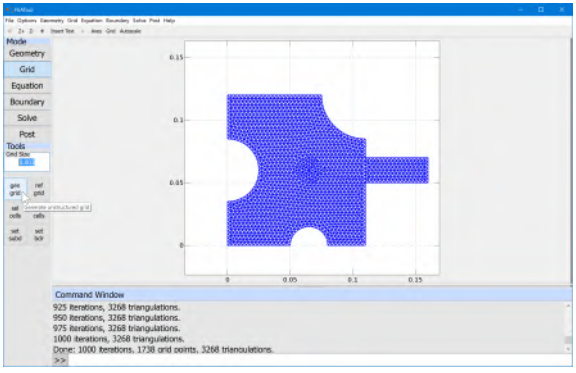
7. Create another rectangle with dimensions (0.065, 0.16, 0.05, 0.07) and circle with radius 0.01 and center (0.065, 0.06) for the second part.



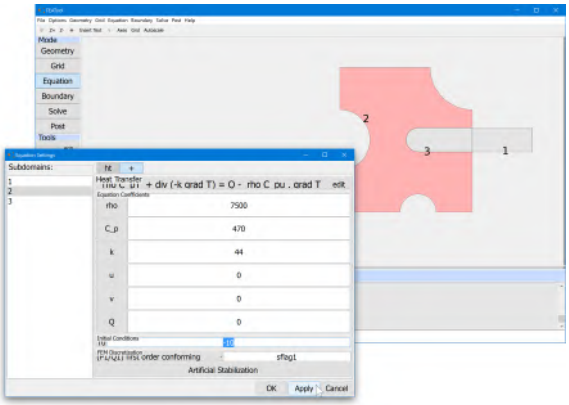
8. In this case the objects should be joined. Click on them so that they are highlighted in red. Alternatively, they can be selected by holding the *Ctrl* key while clicking on the labels **R2** and **E4** in the *Selection* list box (or simply press *Ctrl+a* to select all objects). When the geometry objects are selected, press the **Add geometry objects** button to generate the combined shape.



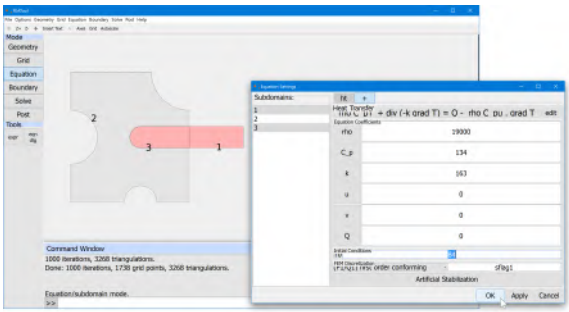
9. Switch to *Grid* mode by click on the corresponding Grid mode toolbar button. Set the target grid size by entering **0.003** in the *Grid Size* edit field or using the slider control, above the grid toolbar buttons. Then click on the **Generate** button Generate to call the automatic grid generation function.



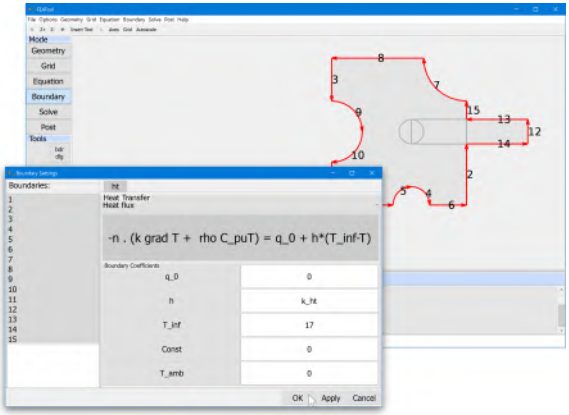
10. Press the Equation mode button in the *Mode* toolbar to switch from *grid* mode to *physics and equation/subdomain* specification mode. In the *Equation Settings* dialog box, first select the subdomain that corresponds to the steel part (here 2), then set the density ρ to **7500**, heat capacity C_p to **470**, and thermal conductivity k to **44** in the corresponding edit fields. Also set the initial temperature T_0 here to -10.



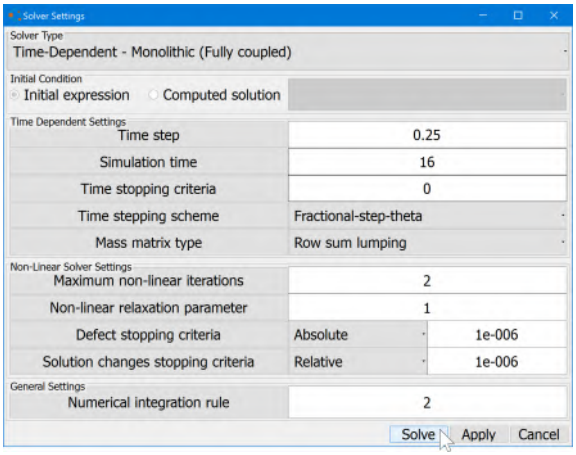
11. Now set the material parameters for the Tungsten part by selecting the other subdomains (1 and 3). Here set the density ρ to **19000**, heat capacity C_p to **134**, thermal conductivity k to **163**, and the initial temperature T_0 to **84**. Press **OK** to finish.



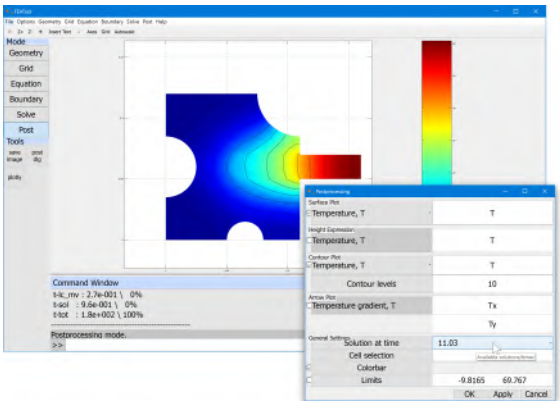
12. Switch to *boundary condition specification* mode by clicking on the `Boundary` mode button. In the *Boundary Settings* dialog box, select the **Heat flux** boundary condition for all the boundaries. Enter **k_ht** for the convective transfer coefficient h (this will use the actual value for the subdomain parameter *k* so that we don't have to enter the values again), and **17** for the surrounding reference temperature T_{inf}.



13. Now that the problem is specified, press the Solve mode button to switch to *solve* mode. Since this is a time dependent study, open the solver settings and select the **Time-Dependent - Monolithic (Fully Coupled)** solver. Set the *Time step* to **0.25**, *Simulation time* to **16**, *Time stopping criteria* to 0, also *Maximum non-linear iterations* to **2**. Then press **Apply** and **Solve** to start the solution process.



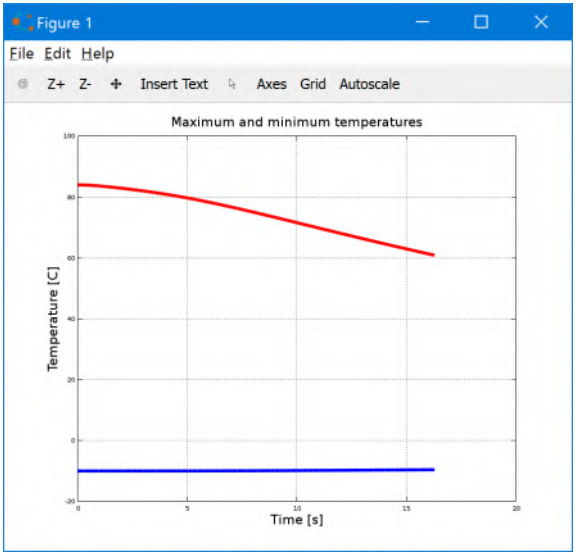
14. After the problem has been solved the temperature at the final time will be shown. To find the time where the maximum temperature is 70 one can study the solution at different times by using the corresponding option in the **Postprocessing Settings** dialog box. Note that both the colorbar and *Limits* field will show the minimum and maximum surface plot value.



15. To see this more clearly we can process the simulation data on the command line by using the **Export** > **FEA Problem Struct To Main Workspace** from the **File** menu. Then switch to the main command window and use the following script commands to see that $T = 70$ is reached at about $t = 11$.

```
tlist = fea.sol.t;
for i=1:numel(tlist)
  T_min(i) = min(fea.sol.u(:,i));
  T_max(i) = max(fea.sol.u(:,i));
end
ix = find(T_max<70);
i1 = ix(1);
i2 = i1 - 1;
s  = ( T_max(i2) - 70 )/( T_max(i2) - T_max(i1) );
t_70 = tlist(i2) + s*( tlist(i1) - tlist(i2) )
u_70 = fea.sol.u(:,i2) + ...
       s*( fea.sol.u(:,i1) - fea.sol.u(:,i2) )

figure
plot( tlist, T_min, 'b-' )
hold on
plot( tlist, T_max, 'r-' )
```

```
grid on
title('Maximum and minimum temperatures')
ylabel('Temperature [C]')
xlabel('Time [s]')
```



7.1.2 Heat Transfer - Shrink Fitting of an Assembly using the CLI

How to set up and solve the thermal shrink fitting problem on the command line interface is illustrated in the ex_heattransfer5 script file which can be found in the FEATool examples directory.

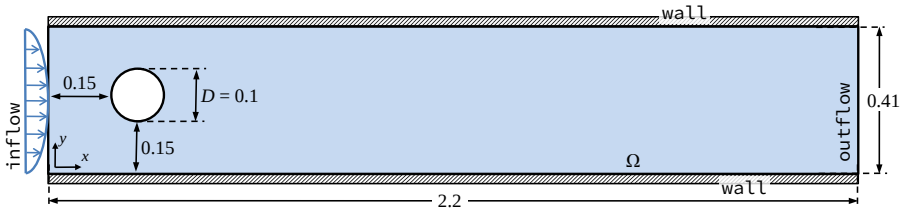
7.1.3 References

[1] Krysl P. A Pragmatic Introduction to the Finite Element Method for Thermal and Stress Analysis. Pressure Cooker Press, USA, 2005.

7.2 Fluid Dynamics - Flow Around a Cylinder

A well known benchmark, test, and validation problem suite for incompressible fluid flows are the DFG cylinder benchmark problems. Although it is not possible to derive analytical solutions to these test cases, accurate numerical solutions to benchmark reference quantities have been established for a number of configurations [3], [4].

The test configuration used in the following places a solid cylinder centered at (*0.2*, *0.2*) with diameter $D = 0.1$ in a $L = 2.2$ by $H = 0.41$ rectangular channel. The fluid density ρ is taken as *1* and the viscosity μ is *0.001*. A fully developed parabolic velocity profile is prescribed at the inlet, $\mathbf{u}_{inflow} = \mathbf{u}(0, y) = 4U_{max}H^{-2}(y(H - y), 0)$, with the maximum velocity $U_{max} = 0.3$. This results in a mean velocity $U_{mean} = 0.2$ and a Reynolds number $Re = \rho U_{mean}D/\mu = 20$ and thus the flow field will be laminar.



As the fluid is considered incompressible the problem is governed by the Navier-Stokes equations. That is,

$$\begin{cases} \rho\left(\dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) - \nabla\cdot(\mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)) + p = \mathbf{F} \\ \nabla\cdot\mathbf{u} = 0 \end{cases}$$

where in this case the time dependent term can be neglected. The benchmark quantities that should be computed include the pressure difference between the front and rear of the cylinder $\Delta p = p(0.15, 0.2) - p(0.25, 0.2)$, and the coefficients of drag c_d and lift c_l, defined as

$$c_d = \frac{2F_d}{\rho U_{mean}^2 D}, \qquad c_l = \frac{2F_l}{\rho U_{mean}^2 D}$$

The drag and lift forces, F_d and F_l, can be computed as

$$F_d = \int_S \left(\mu \frac{\partial \mathbf{u}_\tau(t)}{\partial n} n_y - p n_x\right) dS, \qquad F_l = -\int_S \left(\mu \frac{\partial \mathbf{u}_\tau(t)}{\partial n} n_x + p n_y\right) dS$$

where $\mathbf{u}_\tau$ is the velocity in the tangential direction $\tau = (n_y, -n_x, 0)^T$.

7.2.1 Flow Around a Cylinder using the GUI

This section describes how to set up and solve the Flow Around a Cylinder example with the FEATool graphical user interface (GUI).

1. Start Octave or Matlab, and if you have not run the installation script (which automatically adds the FEATool directory paths at startup) then change your working directory to where your FEATool installation is, for example

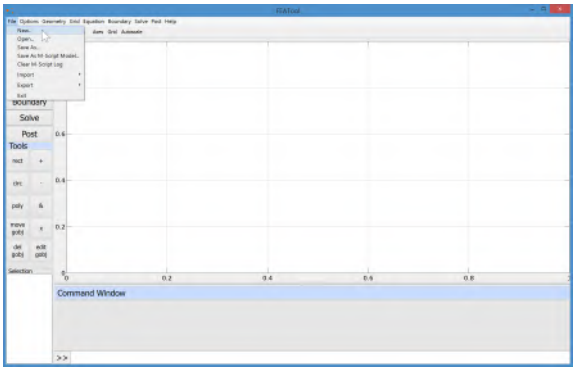
```
cd C:\featool
```

2. In the command window type

```
featool
```

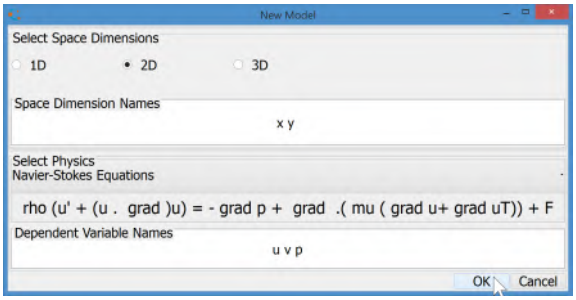
   to start the graphical user interface (GUI).

3. Click on the **New Model** button in the upper horizontal toolbar to clear all data and start defining a new model.

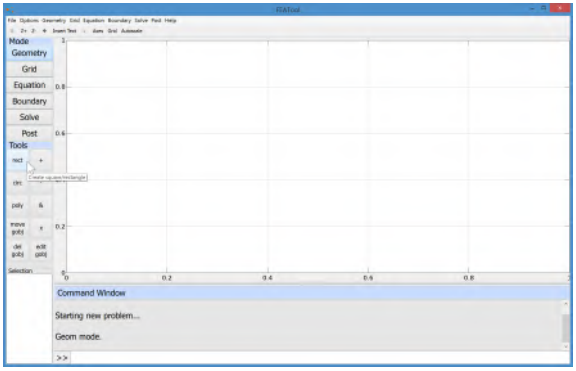


4. In the opened *New Model* dialog box, click on the **2D** radio button in the *Select Space Dimensions* frame, and select **Navier-Stokes Equations** from the *Select Physics* drop-down

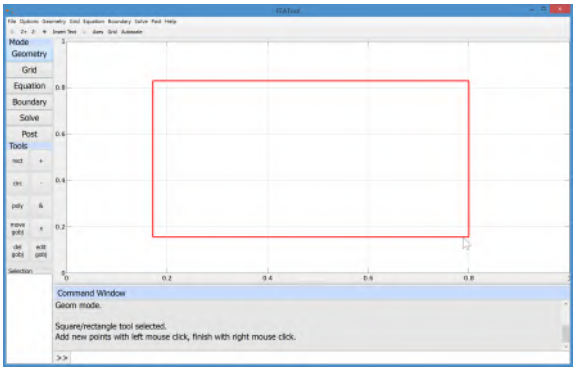
list. Leave the space dimension and dependent variable names to their default values. Finish and close the dialog box by clicking on the **OK** button.



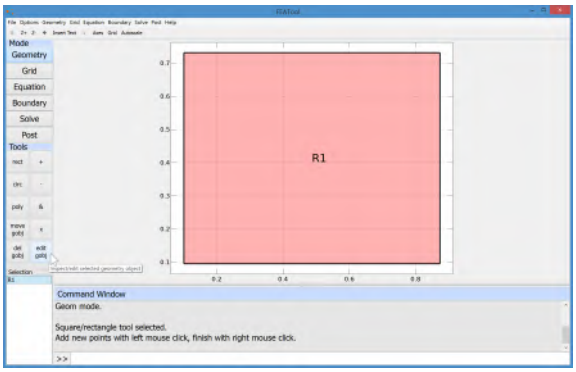
5. To create the outer rectangle, first click on the **Create square/rectangle** button in the left hand side *Tools* toolbar frame.



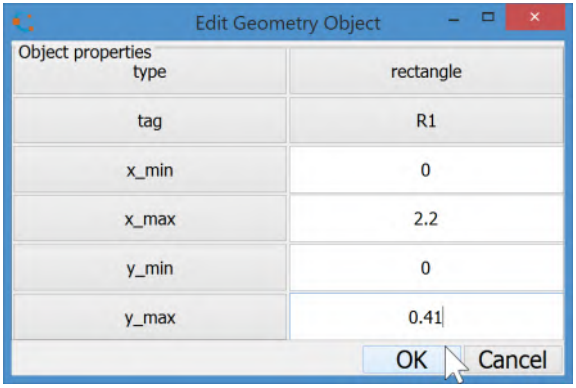
6. Then left click in the main plot axes window, hold the mouse button, and move the mouse pointer to show a rectangle with red outlines.



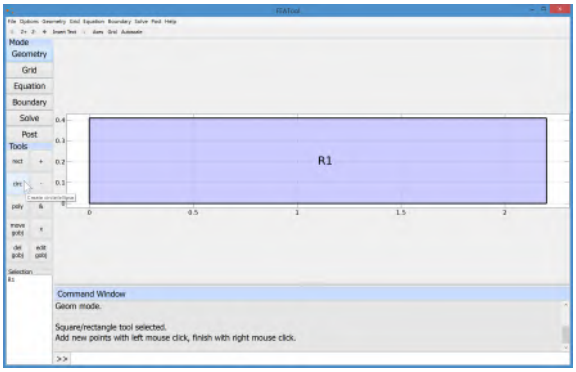
7. Release the mouse button to finalize and create a solid geometry object. The object properties must now be edited to set the correct size and position of the rectangle. To do this, click on the rectangle *R1* to select it and highlight it in red. Then click on the **Inspect/edit selected geometry object** toolbar button



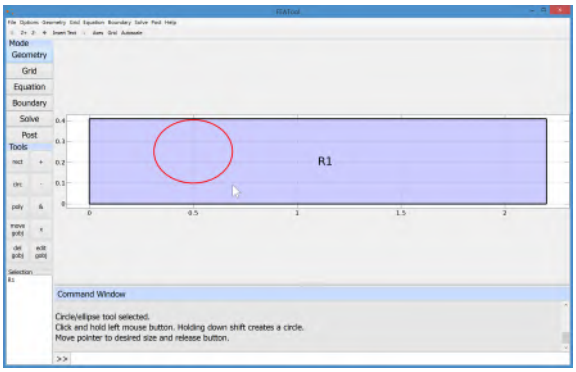
8. In the opened **Edit Geometry Object** dialog box, edit the min and max point coordinates to specify a rectangle with length 2.2 and height 0.41. Finish editing the geometry object and close the dialog box by clicking **OK**.



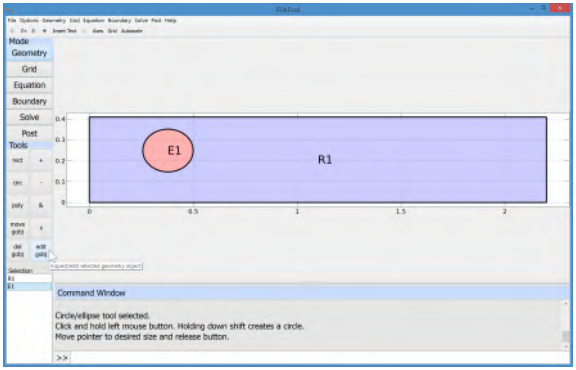
9. To create the inner circle, first left click on the **Create circle/ellipse** button in the left hand side *Tools* toolbar frame.



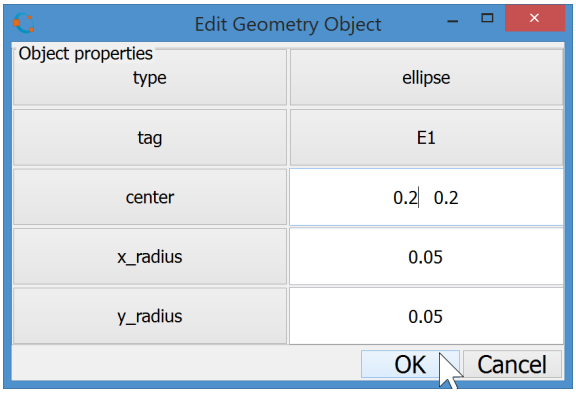
10. Then click and hold the left mouse button anywhere in the main plot axes window, move the mouse pointer to show red outlines of a circle or ellipse. Release the button to finalize and create a solid geometry object.



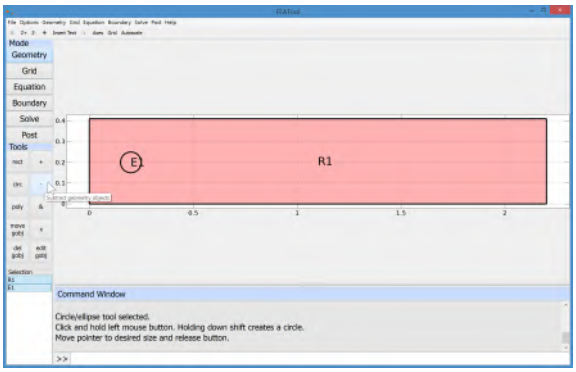
11. The object properties of the *E1* must also be changed to make a circle with radius 0.05 centered at (0.2, 0.2). To do this, click on the *E1* to select it which also highlights it in red. Then again click on the **Inspect/edit selected geometry object** toolbar button



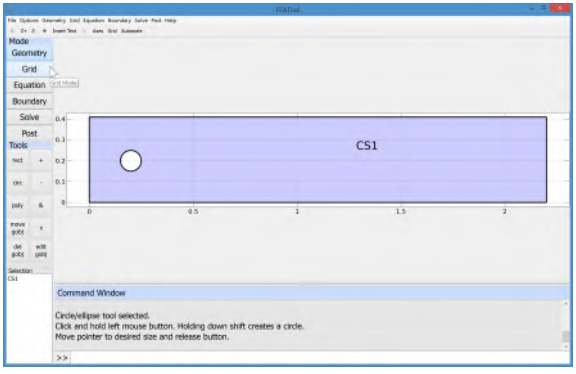
12. In the opened **Edit Geometry Object** dialog box change the center coordinates to **0.2 0.2**, and the *x* and *y* radius **0.05** in the corresponding edit fields. Finish editing *E1* and close the dialog box by clicking **OK**.



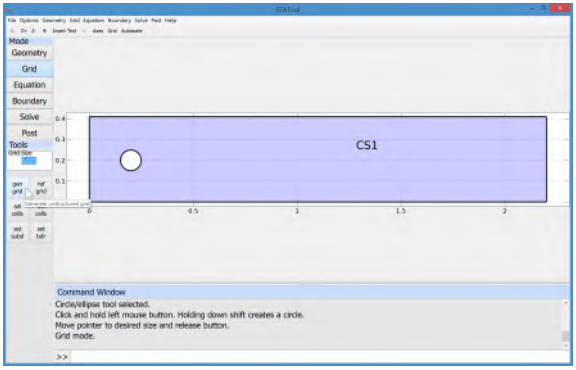
13. To subtract the circle from the rectangle first select both geometry objects by clicking on them so both are highlighted in red. Alternatively, if the circle is obscured by the rectangle they can be selected by holding the *Ctrl* key while clicking on the labels **R1** and **E1** in the *Selection* list box (or simply press *Ctrl+a* to select all objects). When the geometry objects are selected, press the **Subtract geometry objects** button to generate the final geometry shape.



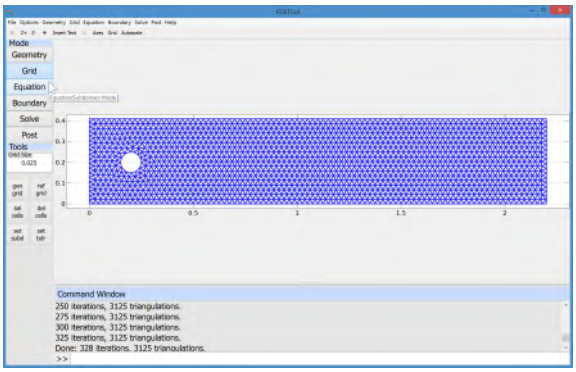
14. Switch to *Grid* mode by click on the corresponding Grid mode toolbar button.



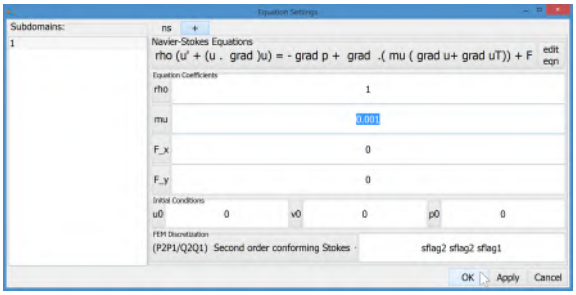
15. Set a target grid size by entering **0.025** in the *Grid Size* edit field above the grid toolbar buttons. Then click on the **Generate** button Generate to call the automatic grid generation function.



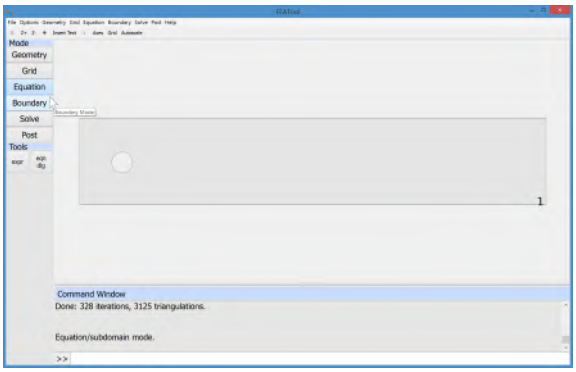
16. Press the Equation mode button in the *Mode* toolbar to switch from *grid* mode to *physics and equation/subdomain* specification mode.



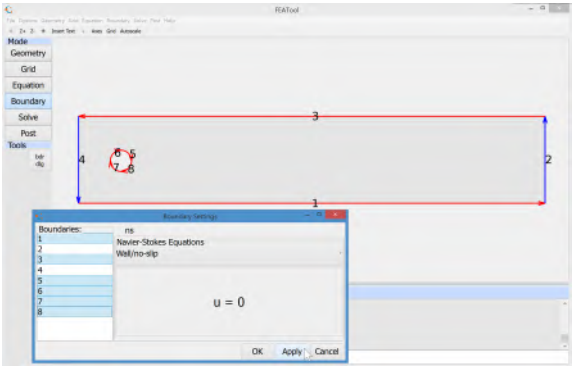
17. In the *Equation Settings* dialog box that automatically opens, set the density ρ to **1** and viscosity μ to **0.001** in the corresponding edit fields. The other coefficients can be left to their zero default values. Press **OK** to finish and close the dialog box.



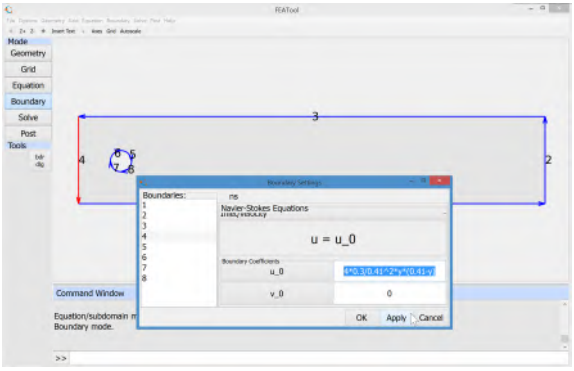
18. Switch to *boundary condition specification* mode by clicking on the Boundary mode button



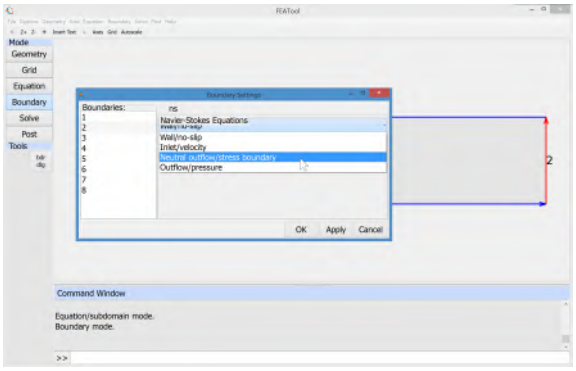
19. In the *Boundary Settings* dialog box, select all boundaries except the right and left ones (here boundaries 1, 3, and 5-8) in the left hand side *Boundaries* list box and choose the **Wall/no-slip** boundary conditions from the drop-down list.



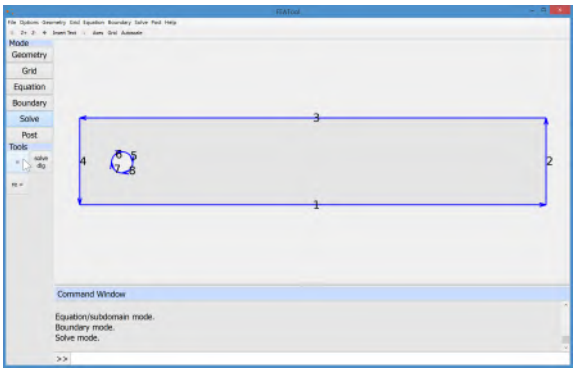
20. Now select the left inflow boundary (here number 4) in the left hand side *Boundaries* list box and choose the **Inlet/velocity** boundary condition from the drop-down list. To specify a parabolic velocity profile enter the expression **4∗0.3/0.41^2∗y∗(0.41-y)** in the edit field for the x-velocity coefficient u_o.



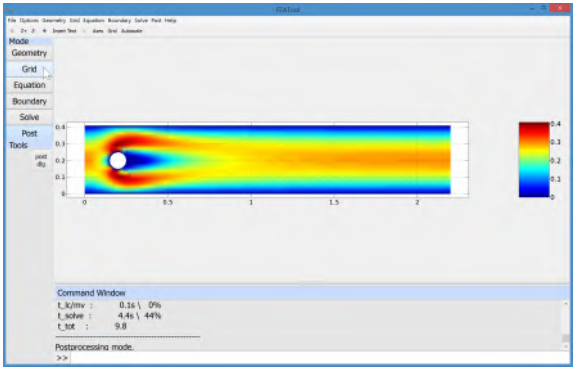
21. Select the right outflow boundary (here number 2) in the left hand side *Boundaries* list box and choose the **Neutral outflow/stress boundary** condition from the drop-down list (alternatively one can prescribe a pressure p_o with the **Outflow/pressure** condition). Finish by clicking the **OK** button.



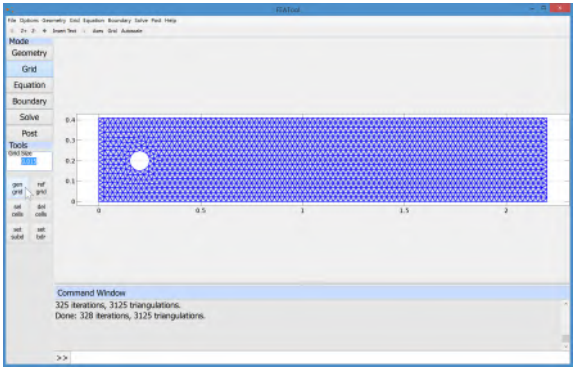
22. Now that the problem is specified, press the Solve mode button to switch to *solve* mode. Then press the button with an *equals* sign = in the *Tools* toolbar frame to call the solver with the default solver settings.



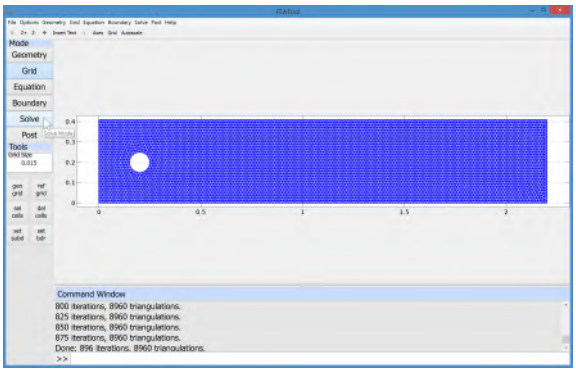
23. After the problem has been solved FEATool will automatically switch to *postprocessing* mode and display the computed solution. The default postprocessing visualization shows the magnitude of the velocity field. Since the solution does not look quite smooth and resolved it might be worthwhile to refine the grid. Press the Grid mode button in the *Mode* toolbar to switch back to *grid* mode again.



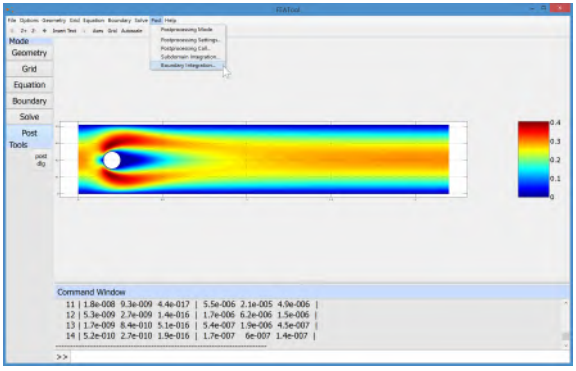
24. Although we could use the uniform grid refinement option, it is usually better to use the automatic grid generation routine with a smaller grid size for better grid quality and approximation of curved boundaries. Set a new target grid size by entering **0.015** in the *Grid Size* edit field above the grid toolbar buttons. Then click on the **Generate** button Generate.



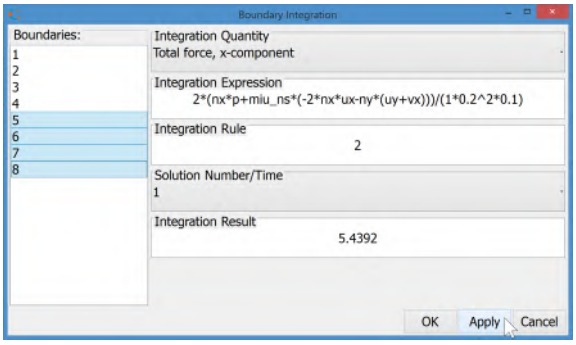
25. Now go back to **Solve** mode and solve the problem again with the refined grid.



26. Now that the solution looks better we can calculate the drag and lift coefficients. To do this select **Boundary Integration...** from the *Post* menu.



27. In the *Boundary Integration* dialog box, select boundaries which make up the circle (5-8) in the left hand side *Boundaries* list box. Then enter the expression for the drag coefficient **2∗(nx∗p+miu_ns∗(-2∗nx∗ux-ny∗(uy+vx)))/(1∗0.2^2∗0.1)** in the *Integration Expression* edit field. Press the **Apply** button to show the result in the lower *Integration Result* frame and in the FEATool terminal window.



28. The lift coefficient can similarly be computed by evaluating the expression **2∗(T_y)/(1∗0.2^2∗0.1)** where **Ty** is the total force, y-component expression (which can be selected in the drop down box), defined by ny∗p+miu_ns∗(-nx∗(vx+uy)-2∗ny∗vy). The magnitudes of the computed drag and lift coefficients are here, 5.4392 and 0.0137, which can be compared with the benchmark reference solutions, that is 5.5795 and 0.010619. The deviation from the reference values suggest that it might be desirable to recompute a solution with an even finer grid, possibly refined around the circle. See for example ex_navierstokes3 where a very adapted benchmark grid is constructed manually.

7.2.2 Flow Around a Cylinder using the CLI

The process to set up and solve the flow around cylinder test problem on the command line interface is illustrated in the ex_navierstokes3 script file which can be found in the examples directory.

7.2.3 Instationary Flow Around a Cylinder

In addition to the stationary test case described above. An instationary benchmark test case is also available (ex_navierstokes6). This test case uses the same geometry but instead applies an inflow condition that varies with time (u_inflow = 6∗sin(pi∗t/8)∗(y∗(0.41-y))/0.41^2) so that 0<Re(t)<100 [5]. Computations with FEATool show that the drag and lift coefficients, and pressure difference between front and rear of the cylinder agrees very well with the reference values [6].

7.2.4 References

[3] Nabh G. On higher order methods for the stationary incompressible Navier-Stokes equations. PhD Thesis 1998; Universität Heidelberg. Preprint 42/98, 1998.

[4] John V, Matthies G. Higher-order finite element discretizations in a benchmark problem for incompressible flows. International Journal for Numerical Methods in Fluids 2001; 37(8):885–903, DOI: 10.1002/fld.195.

[5] John V. Reference values for drag and lift of a two-dimensional time-dependent flow around a cylinder. International Journal for Numerical Methods in Fluids 2004; 44:777-788.

[6] John V, Rang J. Adaptive time step control for the incompressible Navier–Stokes equations. Comput. Methods Appl. Mech. Engrg. 199 (2010) 514–524.

7.3 Fluid Dynamics - Axisymmetric Fluid Flow

The Equation Editing Example - Axisymmetric Fluid Flow example in the quickstart guide shows how to edit predefined equations in the physics modes to model rotationally symmetric or axisymmetric flow in a narrowing pipe.

7.4 Fluid Dynamics - Flow Over a Backwards Facing Step

Flow over a backwards facing step is another classic computational fluid dynamics test problem which has in various forms been used quite extensively for validation of simulation codes. The test problem essentially consists of studying how a fully developed flow profile reacts to a sudden expansion or step in a channel. The expansion will cause a break in the flow and a recirculation or separation zone will form. To measure and compare results the resulting length of the recirculation or separation zone is most often used.

The stationary incompressible Navier-Stokes equations are to be applied corresponding to a Reynold number of $Re = 389$. The size domain is chosen according to the data below. The inlet velocity function is given as $u_{in}(0,y) = 4u_{max}h^{-2}(y(h-y),0)$ where h is the channel height, and maximum velocity $u_{max} = 1$ No-slip zero velocity conditions are applied to all walls and a suitable outflow condition must also be applied. The reference recirculation zone length found in reference [7] and [8] is estimated to be 7.93 length units.

7.4.1 Flow over a Backwards Facing Step with the external OpenFOAM CFD Solver

This section describes how to set up and solve the Flow over a Backwards Facing Step example with the FEATool graphical user interface (GUI) and using the external OpenFOAM CFD Solver.

1. Start Octave or Matlab, and if you have not run the installation script (which automatically adds the FEATool directory paths at startup) then change your working directory to where

your FEATool installation is, for example

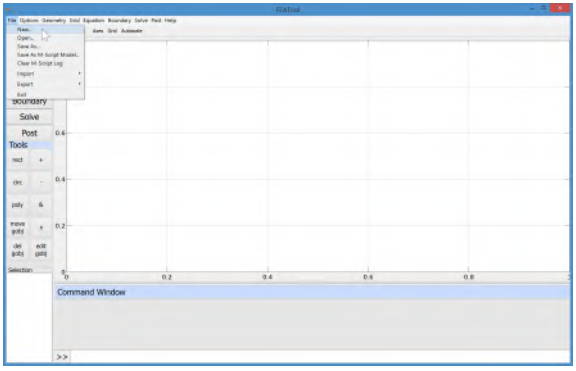
```
cd C:\featool
```

2. In the command window type

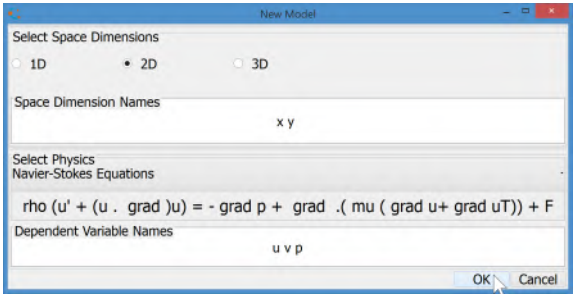
```
featool
```

to start the graphical user interface (GUI).

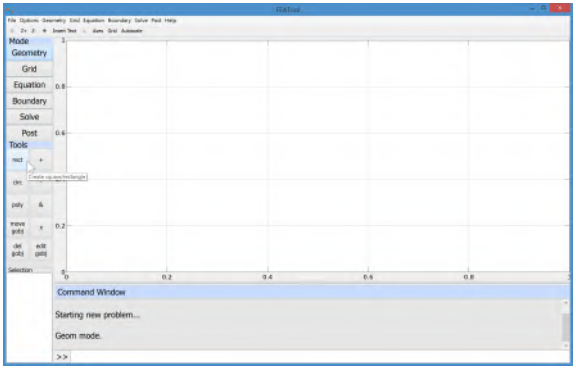
3. Click on the **New Model** button in the upper horizontal toolbar to clear all data and start defining a new model.



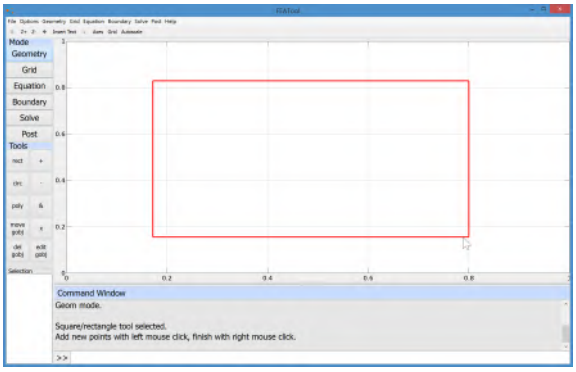
4. In the opened *New Model* dialog box, click on the **2D** radio button in the *Select Space Dimensions* frame, and select **Navier-Stokes Equations** from the *Select Physics* drop-down list. Leave the space dimension and dependent variable names to their default values. Finish and close the dialog box by clicking on the **OK** button.



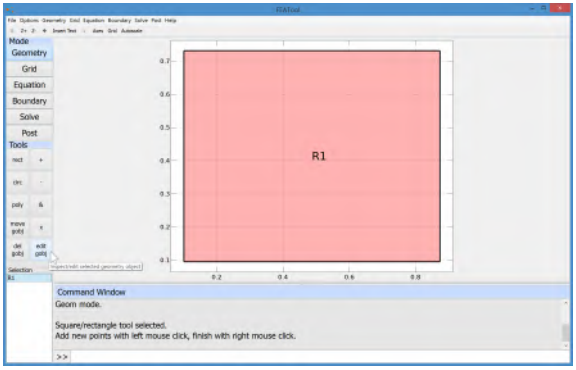
5. To create the outer rectangle, first click on the **Create square/rectangle** button in the left hand side *Tools* toolbar frame.



6. Then left click in the main plot axes window, hold the mouse button, and move the mouse pointer to show a rectangle with red outlines.

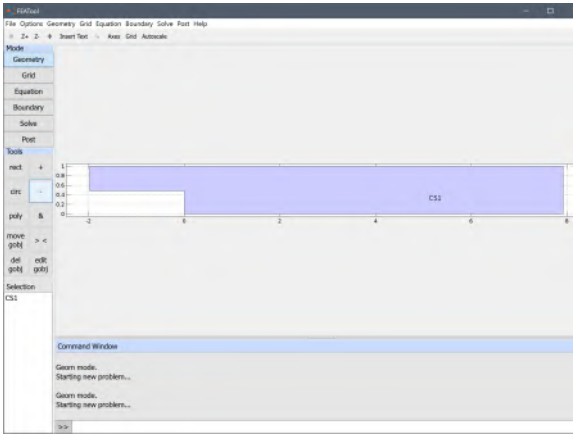


7. Release the mouse button to finalize and create a solid geometry object. The object properties must now be edited to set the correct size and position of the rectangle. To do this, click on the rectangle *R1* to select it and highlight it in red. Then click on the **Inspect/edit selected geometry object** toolbar button



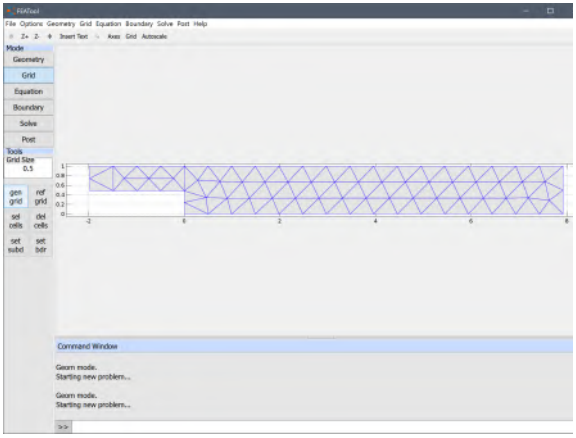
8. In the opened **Edit Geometry Object** dialog box, edit the min and max point coordinates to specify a rectangle with *xmin -1.9802*, *xmax 7.9802*, *ymin 0*, and *ymax 1*. Finish editing the geometry object and close the dialog box by clicking **OK**.

9. Repeat the steps to create another smaller rectangle with *xmin -1.9802*, *xmax 0*, *ymin 0*, and *ymax 0.4851*. Subtract the rectangles from each other by selecting both rectangles and clicking on the button.

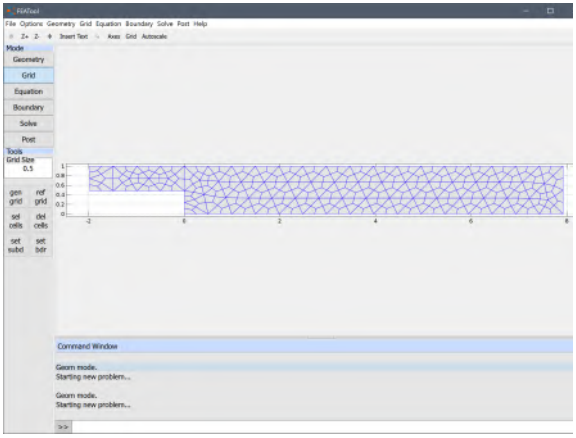


10. Switch to *Grid* mode by click on the corresponding Grid mode toolbar button.

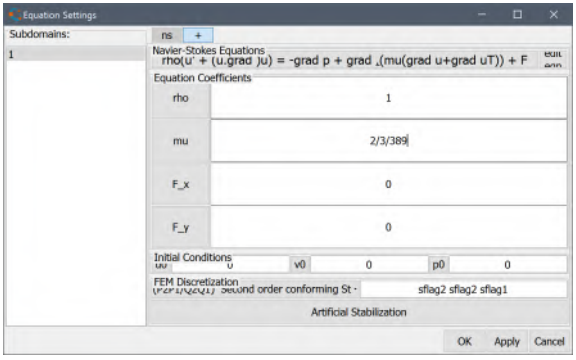
11. Set a target grid size by using the *Grid Size* slider control, or alternatively enter a specific grid size value manually in the corresponding edit field above. Then click on the **Generate** button Generate to call the automatic grid generation function.



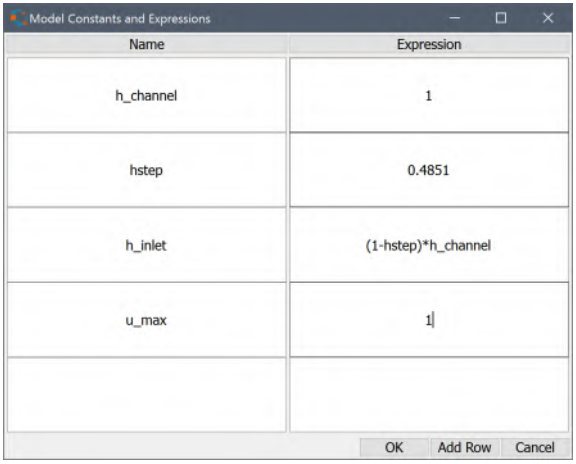
12. Although both FEATool and the *OpenFOAM CFD* solvers support all grid cell types, it is sometimes desirable to work with structured quadrilateral and hexahedral cells. To convert to quadrilateral cells, select **Convert Grid Cells** from the **Grid** menu and apply grid smoothing by using the corresponding **Grid Smoothing** menu options.



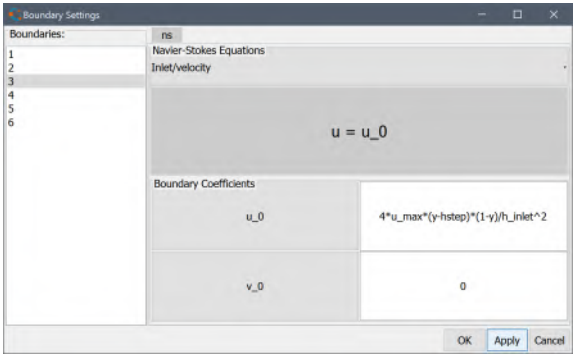
13. Press the `Equation` mode button in the *Mode* toolbar to switch from *grid* mode to *physics and equation/subdomain* specification mode. In the *Equation Settings* dialog box that automatically opens, set the density ρ to **1** and viscosity μ to **2/3/389** in the corresponding edit fields. The other coefficients can be left to their zero default values. Press **OK** to finish and close the dialog box.



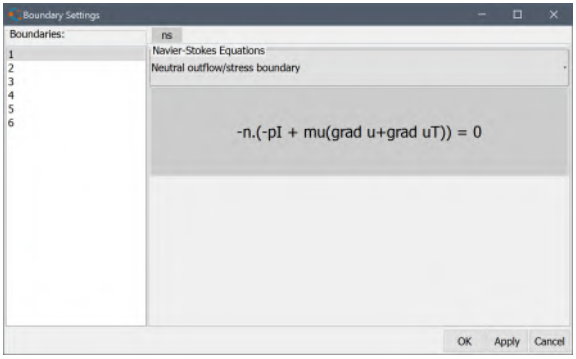
14. Click on the **Model Constrains and Expressions** and enter the constants and expressions shown below.



15. Switch to *boundary condition specification* mode by clicking on the `Boundary` mode button. Select the left inflow boundary (here number 3) in the left hand side *Boundaries* list box and choose the **Inlet/velocity** boundary condition from the drop-down list. When using the default built-in solver enter **4∗u_max∗(y-hstep)∗(1-y)/h_inlet^2**, in the edit field for the x-velocity coefficient u_o, to prescribe a parabolic laminar inlet profile, or when using OpenFOAM enter **3/2∗u_max** to specify a constant mean velocity.

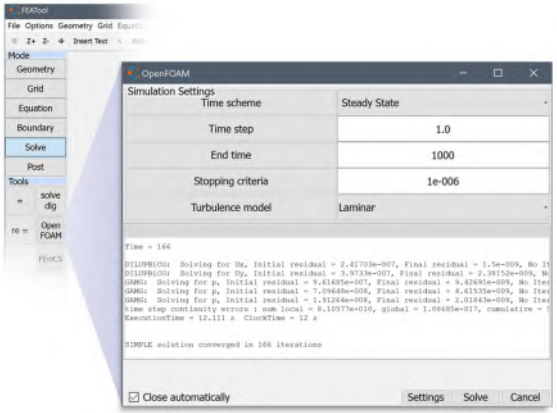


16. Select the right outflow boundary (here number 1) in the left hand side *Boundaries* list box and choose the **Outflow/pressure** boundary condition from the drop-down list. Finish by clicking the **OK** button.

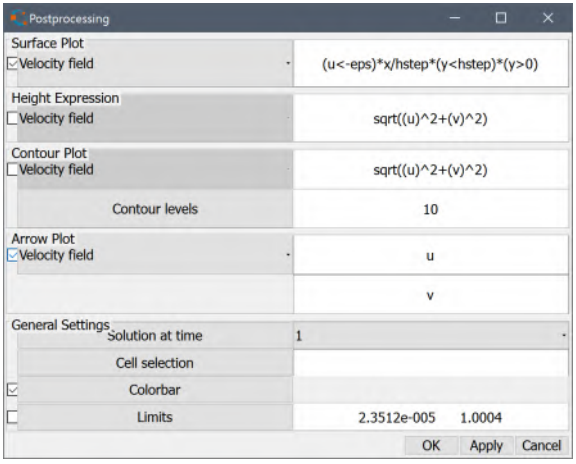


17. Now that the problem is specified, press the Solve mode button to switch to *solve* mode. To use the default built-in solver, press the button with an *equals* sign =. Alternatively, press the button labeled **OpenFOAM** to open the OpenFOAM CFD solver settings and control panel.

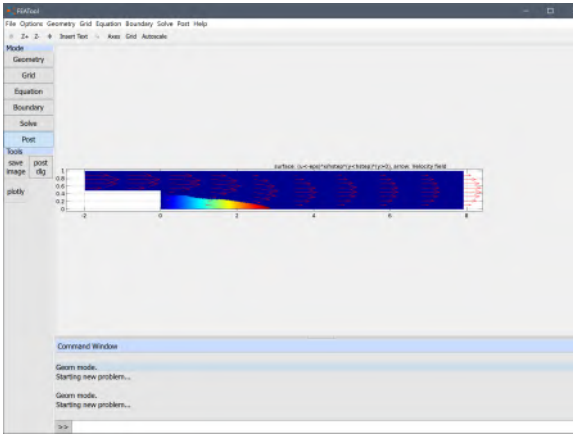
18. In the *OpenFOAM* solver settings dialog box, use the default settings, and press the **Solve** button to call the external *OpenFOAM* solver. When the solver is done the computed solution will automatically be imported, and FEATool will switch to Postprocessing mode.



19. To see the recirculation zone clearly, enter the expression for the normalized recirculation zone length __(u<-eps)∗x/hstep∗(y<hstep)∗(y>0)__ in the *Surface Plot* expression edit field. The **Arrow Plot** option can also be used to visualize the flow field.



20. The resulting solution shows a recirculation zone length of about 6 length units which can be improved with a finer and better grid.



7.4.2 Flow over a Backwards Facing Step using the CLI

The process to set up and solve the flow over a backwards facing step test problem on the command line interface is illustrated in the ex_navierstokes4 script file which can be found in the examples directory.

7.4.3 References

[7] P.M. Gresho and R.L. Sani, Incompressible Flow and the Finite Element Method, Volume 1 & 2, John Wiley & Sons, New York, 2000.

[8] A. Rose and B. Simpson: Laminar, Constant-Temperature Flow Over a Backward Facing Step, 1st NAFEMS Workbook of CFD Examples, Glasgow, UK, 2000.

7.5 Structural Mechanics - Thin Plate with Hole

An example of setting up and solving the Thin Plate with Hole example with the FEATool graphical user interface (GUI) is described in the FEATool quickstart guide Structural Mechanics Example - Thin Plate with Hole.

7.6 Structural Mechanics - 3D Deflection of a Bracket

In this section a three dimensional model of a bracket with a hole will be described. The vertical side of the bracket is fixed to a wall (thus zero displacement boundary conditions are suitable here), while the outer end of the bracket is loaded with a negative force of -10000 in the z-direction. The resulting displacements will be studied together with creating a displacement plot visualizing the deformation.

7.6.1 3D Deflection of a Bracket using the GUI

This section describes how to set up and solve the bracket stress problem with the FEATool graphical user interface (GUI).

1. Start Octave/Matlab, and if you have not run the installation script (which automatically adds the FEATool directory paths at startup) then change your working directory to where your FEATool installation is, for example

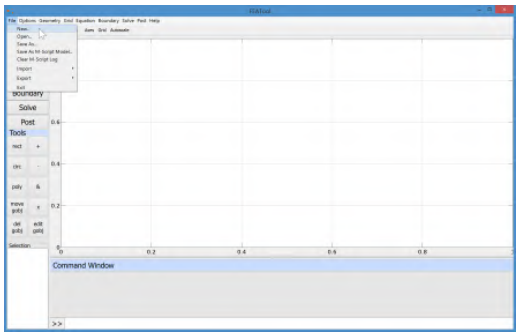
   ```
   cd C:\featool
   ```

2. In the command window type

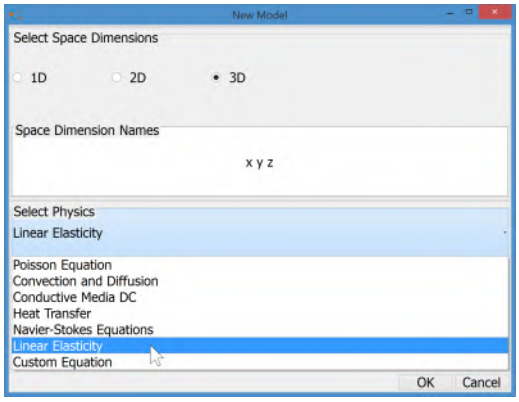
   ```
   featool
   ```

   to start the graphical user interface (GUI).

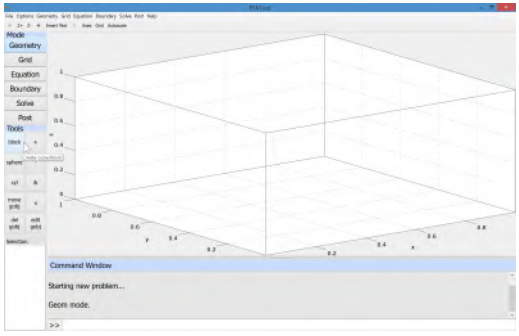
3. Click on the **New Model** button in the upper horizontal toolbar to clear all data and start defining a new model.



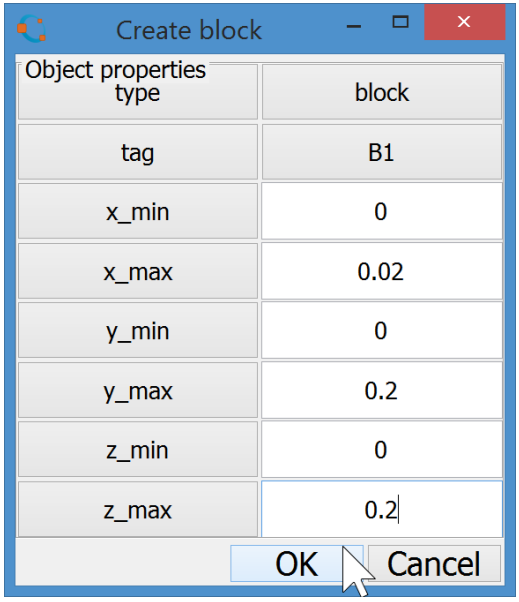
4. In the opened *New Model* dialog box, click on the **3D** radio button in the *Select Space Dimensions* frame, and select **Linear Elasticity** from the *Select Physics* drop-down list. Leave the space dimension and dependent variable names to their default values. Finish and close the dialog box by clicking on the **OK** button.



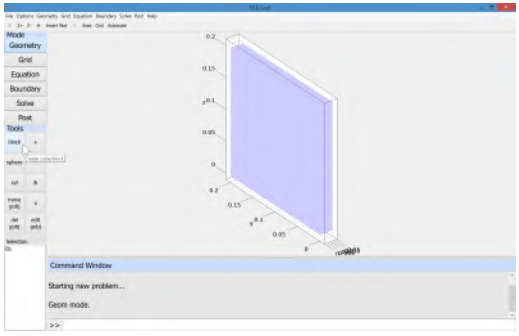
5. To create the first block for the bracket, first click on the **Create cube/block** button in the left hand side *Tools* toolbar frame.



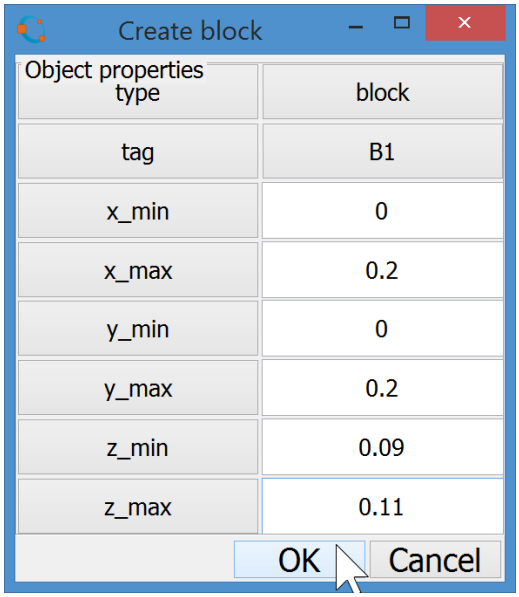
6. In the open *Create block* dialog box, set the extents of the block to [0 0.02], [0 0.2], [0 0.2], for the *x*, *y*, and *z* min and max coordinates. Press **OK** to finish and generate the first block.



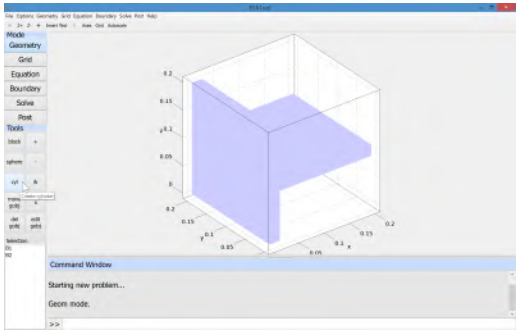
7. Click on the **Create cube/block** button again to create the second section.



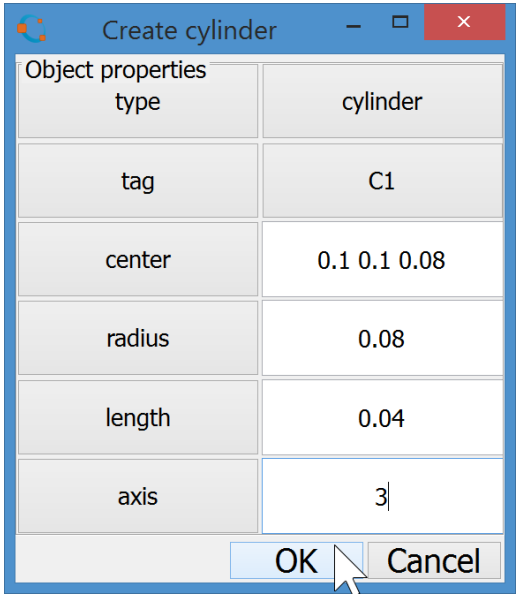
8. In the open *Create block* dialog box, now set the extents of the block to [0 0.2], [0 0.2], [0.09 0.11], for the *x*, *y*, and *z* min and max coordinates. Press **OK** to finish and generate the second block.



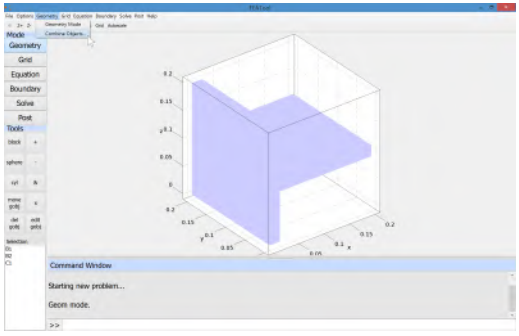
9. To create the cylindrical section click on the **Create cylinder** button .



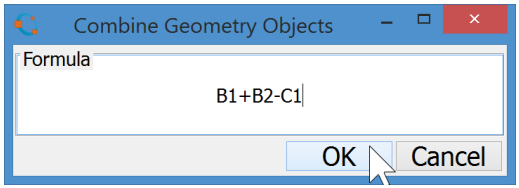
10. In the open *Create block* dialog box, now set the cylinder center to [0.1 0.1 0.08], the radius to 0.08, the length to 0.04, and alignment axis to 3 (pointing in the z-direction). Press **OK** to finish and generate the cylinder.



11. To create the final geometry a composite object must be generated. To do this select **Combine Objects...** from the **Geometry** menu.

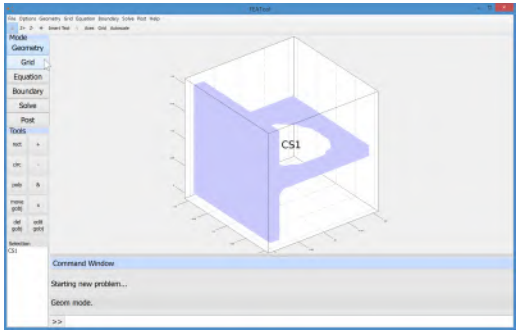


12. Enter the formula **B1+B2-C1** in the opened dialog box and finish by clicking **OK**.

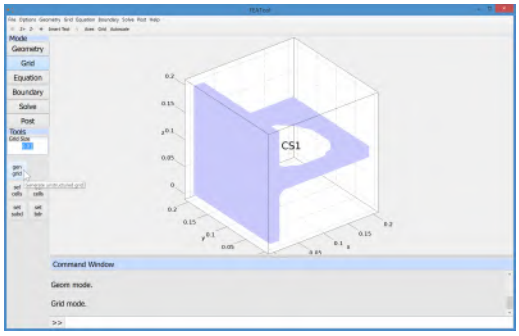


13. Switch to *Grid* mode by clicking on the corresponding

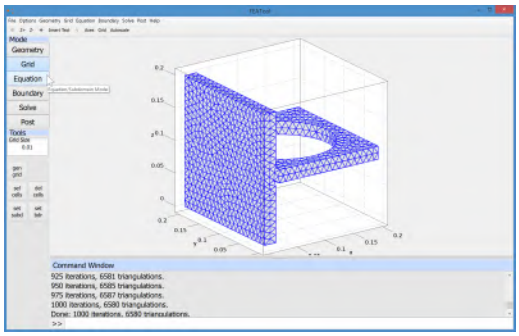
mode toolbar button.



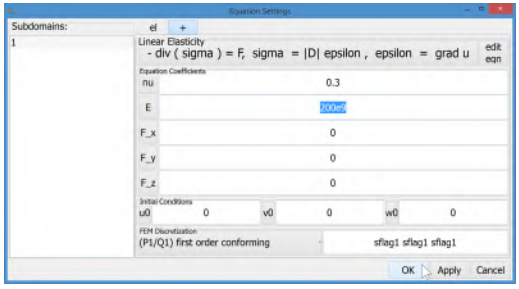
14. In *grid* mode, first set the **Grid Size** to *0.01* in the left toolbar edit field (this is half of the thickness so that a minimum of two cells are generated in each direction). Then click on the **Generate** button Generate to call the grid generation function which automatically generates a grid of tetrahedra. Note that this can take some time depending on your system configuration.



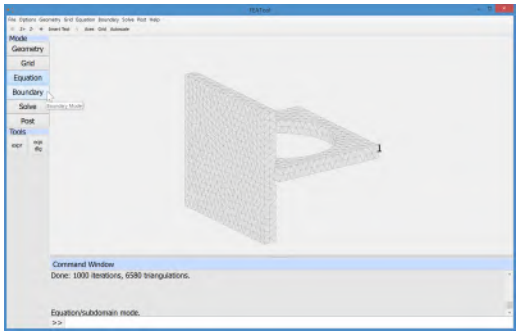
15. Press the Equation mode button in the *Mode* toolbar to switch from *grid* mode to *physics and equation/subdomain* specification mode.



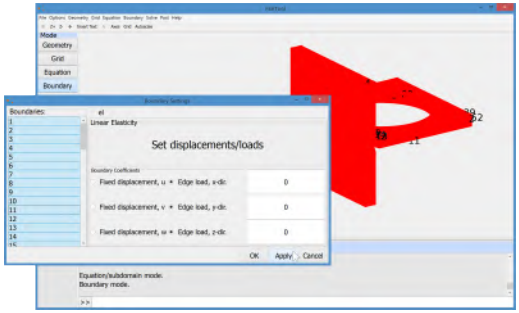
16. In the *Equation Settings* dialog box that automatically opens, set Poisson's ratio ν to **0.3** and the modulus of elasticity E to **200e9** in the corresponding edit fields. The other coefficients can be left to their default values. Press **OK** to finish and close the dialog box.



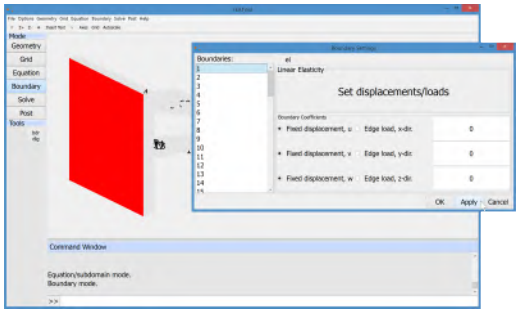
17. Switch to *boundary condition specification* mode by clicking on the Boundary mode button



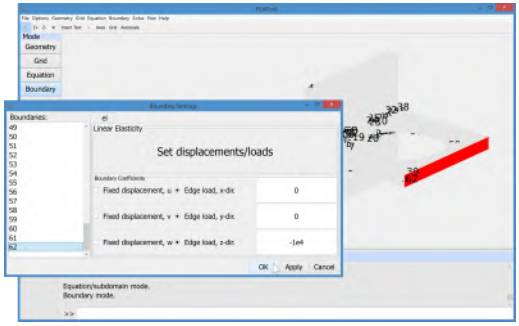
18. In the *Boundary Settings* dialog box, first select all boundaries (this can be done by pressing *Ctrl+a* after clicking in the *Boundaries* list box) and set all conditions to **Edge loads** with a value of zero **0**. Press **Apply** to save these changes.



19. Now select the left vertical boundary (in this case boundary number *1*) and select **Fixed displacement** in all directions. Press **Apply** again to save the changes.

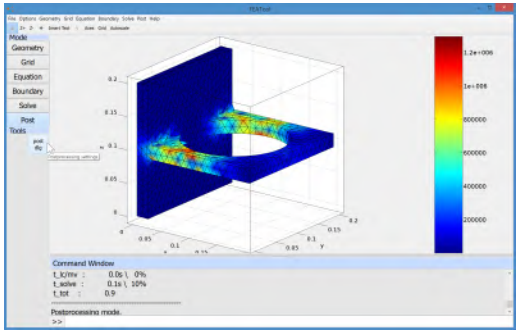


20. Finally select the far right boundary (in this case boundary number *62*) and select set an edge load in the z-direction with value -1e4. Press **OK** to finish prescribing boundary conditions.

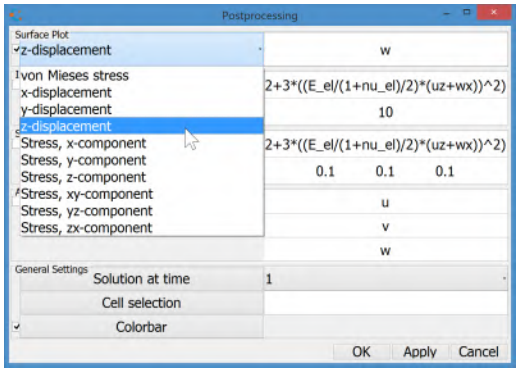


21. Now that the problem is specified, press the Solve mode button to switch to *solve* mode. Then press the button with an *equals* sign = in the *Tools* toolbar frame to call the solver with the default solver settings.

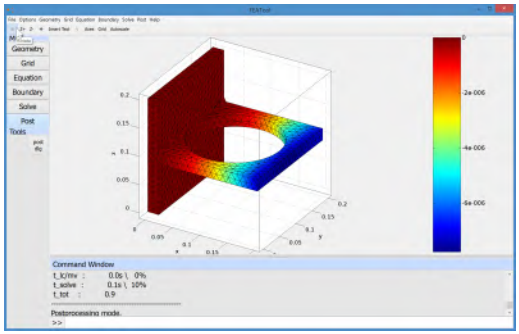
22. After the problem has been solved FEATool will automatically switch to *postprocessing* mode and display the computed von Mieses stress. To change the plot, open the post-processing settings dialog box by clicking on the **Postprocessing settings** button in the *Tools* toolbar frame.



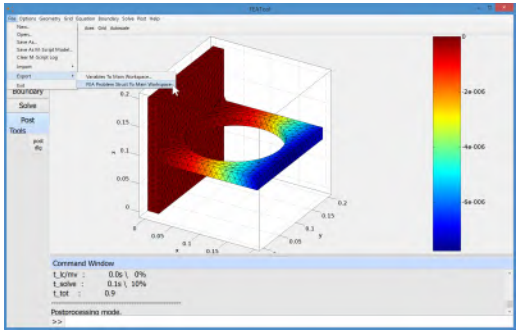
23. In the Postprocessing settings dialog box choose to plot the **z-displacement** as a surface plot.



24. Three dimensional models can be rotated by using the **Rotate** button.

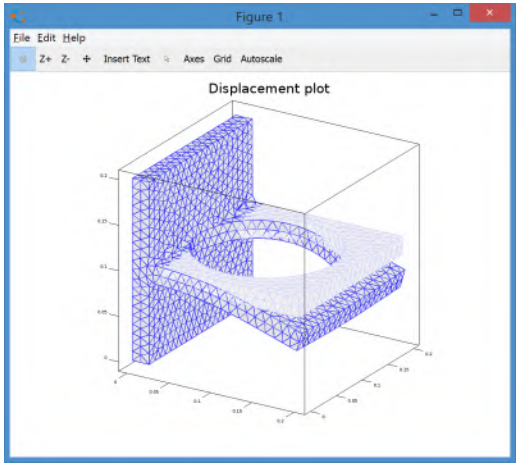


25. To create a displacement plot the model must be processed on the command line. To export the model choose **Export**, **FEA Problem Struct To Main Workspace** from the **File Menu**.



26. The following code scales and evaluates the displacements in the grid points and creates a distorted grid for visualization of the deformation.

```
scaling = 5e3;
dp = zeros(size(fea.grid.p));
for i=1:3
  dp(i,:) = scaling*evalexpr( fea.dvar{i}, ...
              fea.grid.p, fea );
end
fea_disp = fea;
fea_disp.grid.p = fea_disp.grid.p + dp;

f = figure;
left_cells = selcells(fea,['x>',num2str(1.5*0.02)]);
plotgrid( fea, 'facecolor', [.95 .95 .95], ...
               'edgecolor', [.8 .8 1], ...
               'selcells', left_cells, ...
               'parent', gca )
hold on
plotgrid( fea_disp, 'parent', gca )

title( 'Displacement plot' )
view( [30 20] )
```



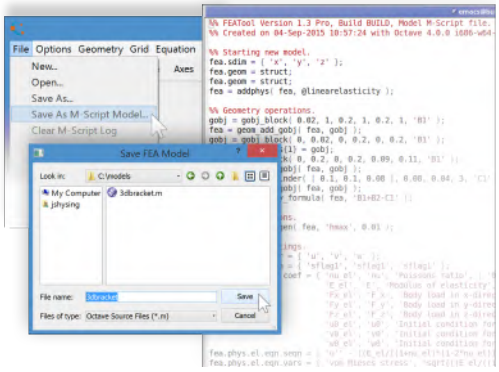
7.6.2 3D Deflection of a Bracket using the CLI

The process to set up and solve the bracket stress test problem on the command line interface is illustrated in the ex_linearelasticity2 script file which can be found in the examples directory.

7.6.3 Parametric Study of Deflection of a Bracket

Parametric studies can be very useful tools to clearly see how a model will behave under varying conditions. The following section presents a parametric study of the bracket model varying both the geometry, in this case plate thickness, and also the applied load force.

1. The first step in creating a parametric model is to save the model as an m-script model file. This can be done by selecting **Save as M-Script Model** from the **File** menu.



2. Open the file in your favorite editor. It is now possible to see exactly which steps and functions were used in creating the model. The first change to make is to add vectors of parameters at the top of the file

```
thickness = [ 0.02 0.015 0.01 0.008 ];
loads     = [ 1e4 2e4 1e5 1e6 ];
```

3. The second change is to replace the geometry and grid operations with the following code so that the thickness of the second block will be varied with the iteration counter *i*

```
for i=1:length(thickness)

% Geometry operations.
fea.geom = struct;
gobj = gobj_block( 0, 0.02, 0, 0.2, 0, 0.2, 'B1' );
fea  = geom_add_gobj( fea, gobj );
gobj = gobj_block( 0, 0.2, 0, 0.2, ...
             0.1-thickness(i)/2, 0.1+thickness(i)/2, 'B1' );
fea  = geom_add_gobj( fea, gobj );
gobj = gobj_cylinder( [ 0.1, 0.1, 0.08 ], 0.08, 0.04,3,'C1');
fea  = geom_add_gobj( fea, gobj );
fea  = geom_apply_formula( fea, 'B1+B2-C1' );

% Grid generation.
fea.grid = gridgen( fea, 'hmax', thickness(i)/2 );
```

4. As second loop can now be introduced where the varying load is extracted and prescribed by using a constant variable. The equation settings can be kept from the saved model.

```
for j=1:length(loads)

% Constants and expressions.
fea.expr = { 'load', loads(j) };

% Equation settings.
fea.phys.el.dvar = { 'u', 'v', 'w' };
fea.phys.el.sfun = { 'sflag1', 'sflag1', 'sflag1' };
fea.phys.el.eqn.coef = { 'nu_el', ',␣'',␣{␣'0.3' };
...
```

   Two loops have now been set up around the model, the first creating a geometry with varying thickness, and the second will prescribe the varying load. Note that setting up the loops in this way allows reusing the grid in the inner loop and saving computational time.

5. When the grid changes the boundary numbering may also change. The static boundary condition prescription must therefore be replaced with one that identifies the correct

boundaries. This can be achieved in the following way

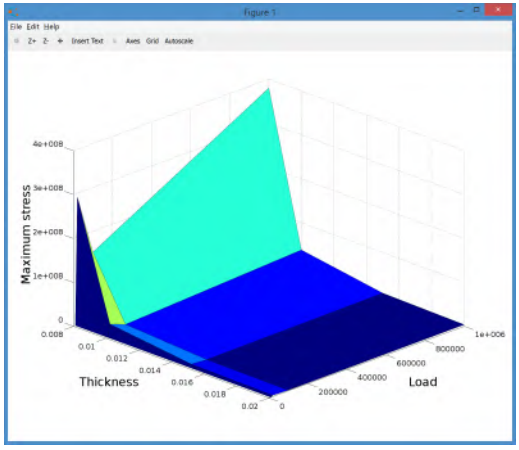
```
% Boundary conditions.
n_bdr = max(fea.grid.b(3,:));
ind_fixed = findbdr( fea, 'x<0.005' );
ind_load  = findbdr( fea, 'x>0.19'  );
fea.phys.el.bdr.sel = ones(1,n_bdr);

% Define Homogeneous Neumann BC type.
n_bdr  = max(fea.grid.b(3,:));
bctype = num2cell( zeros(3,n_bdr) );

% Set Dirichlet BCs for the fixed boundary.
[bctype{:,ind_fixed}] = deal( 1 );
fea.phys.el.bdr.coef{1,end-2} = bctype;

% Define all zero BC coefficients.
bccoef = num2cell( zeros(3,n_bdr) );

% Apply negative z-load to outer edge.
[bccoef{3,ind_load}] = deal( '-load' );

% Store BC coefficient array.
fea.phys.el.bdr.coef{1,end} = bccoef;
```

6. When the problem is fully specified we can call the solver, apply postprocessing to calculate the maximum stress, and close the loops

```
% Solver call.
fea = parsephys( fea );
fea = parseprob( fea );
fea.sol.u = solvestat( fea );

% Postprocessing.
s_vm = fea.phys.el.eqn.vars{1,2};
vm_stress = evalexpr( s_vm, fea.grid.p, fea );
max_vm_stress(i,j) = max(vm_stress);

end,end
```

7. Finally visualize the all computed maximum stresses at once as

```
% Visualization.
[t,l] = meshgrid(thickness,loads);
surf( t, l, max_vm_stress, log(max_vm_stress) )
xlabel( 'Thickness' )
ylabel( 'Load' )
zlabel( 'Maximum_stress' )
view( 45, 30 )
```



The complete parameter study model can be found as the ex_linearelasticity3 script file in the examples directory.

7.7 Electromagnetics - Electrostatic Spherical Capacitor

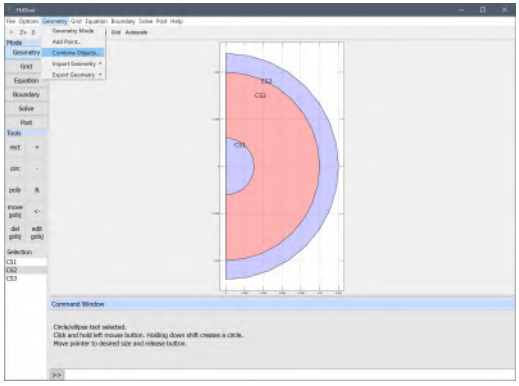
Model example of an axisymmetric electrostatic spherical capacitor compared with analytical solutions.

1. Start a new model and select *Axi 2D* and *Electrostatics*.

2. Start in *Geometry* mode and create three circles all centered at *0 0* but with radii *0.003*, *0.01*, and *0.012*.

3. Create three rectangles on the left side of the symmetry axis ($r<0$) so that they cover the left side of the circles (extending a somewhat above, below and to the left).

4. One by one subtract the rectangles from the corresponding circles so that only half circles on right side half plane remain.



5. Switch to *Grid* mode by clicking on the corresponding Grid mode toolbar button, and generate a grid with a the subdomain *Grid Size* equal to *0.001*.

6. Press the Equation mode button to switch from *grid* mode to *physics and equation/subdomain* specification mode. In the *Equation Settings* dialog box that automatically opens, select all *Subdomains* (1-3) and enter the following coefficients in the *Equation Settings* dialog box

Name	Expression
Permittivity, ε	sigma+epsr∗eps0/tscale
Polarization, x-component, P_x	0
Polarization, y-component, P_y	0
Space charge density, ρ	rho/tscale

7. Select *(P2/Q2) second order conforming* in the *FEM Discretizations* drop down box to icrease the spatial discretization accuracy.

8. Open the *Model Constants and Expressions* dialog box and enter the following expressions (note that expressions for several subdomains are separated with one or more spaces)

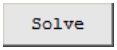
Name	Expression
r1	0.003
r2	0.01

r3	0.012
sigma	6e7 0 6e7
eps0	8.85e-12
epsr	1 3.9 1
tscale	1e-17
rho	q0∗3/4/pi/(r3^3-r2^3) 0 -q0∗3/4/pi/r1^3
q0	6e-11

9. Switch to *boundary condition specification* mode by clicking on the

mode button and select the following boundary conditions

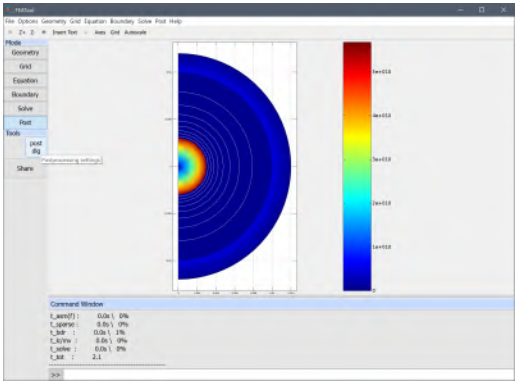
   - *Ground/antisymmetry V=0* for the outer boundaries (3 and 4).
   - *Insulation/symmetry* for all symmetry boundaries *r=0* (1,2,5,6, and 7).

10. Now that the problem is specified, press the 

mode button to switch to *solve* mode. Then press the button with an *equals* sign

to solve the problem.

11. After the problem has been solved, FEATool automatically changes to Postprocessing mode and plots the electric potential *V*.



12. To verify the solution, choose the *Subdomain Integration...* option in the *Post* menu, and integrate the expression *eeng = eps0∗epsr∗(Vr^2+Vz^2)∗pi∗r* over all subdomains. The capacitance can then be calculated as *q0^2/(2∗eeng)*.

13. An alternative way to calculate the capacitance is *q0/(Vmax - Vmin)* where *Vmax* and *Vmin* can be found by using the *Max Min Evaluation...* option from the *Post* menu.

14. Compare the computed capacitances with the analytical expression *4∗pi∗epsr∗eps0/(1/r1 - 1/r2)* which in this example should be equal to *1.8588e-12*.

7.7.1 Electrostatic Spherical Capacitor using the CLI

The process to set up and solve the magnetostatics problem on the command line interface is illustrated in the ex_electrostatics2 script file which can be found in the examples directory.

7.8 Electromagnetics - Magnetic Field Around a Horseshoe Magnet

Magnetostatics example for calculating the magnetic field around a horseshoe magnet.

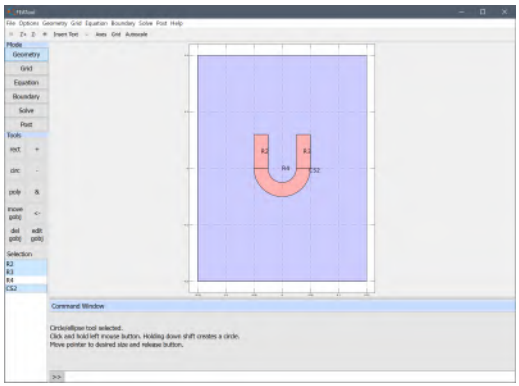
1. Start a new model and select *2D* and *Magnetostatics*.

2. In *Geometry* mode first create two circles, centered at *0 0* with radii *0.05* and *0.025*.

3. Create four rectangles with dimensions according to the table below

	R1	**R2**	**R3**	**R4**
x_min	-0.06	-0.05	0.025	-0.15
x_max	0.06	-0.025	0.05	0.15
y_min	0	0	0	-0.2
y_max	0.06	0.06	0.06	0.2

4. Subtract rectangle *R1* and the smaller circle *C2* from the larger one *C1* using the formula **C1-C2-R1** in the *Combine Objects...* option of the *Geometry* menu.



5. Switch to *Grid* mode by clicking on the corresponding `Grid` mode toolbar button, and generate a grid with a the subdomain *Grid Size* equal to *0.01*.

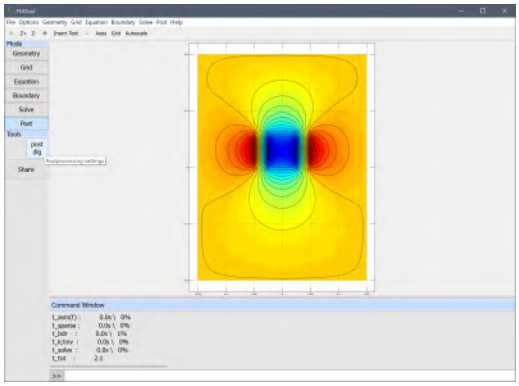
6. Press the `Equation` mode button to switch from *grid* mode to *physics and equation/subdomain* specification mode. In the *Equation Settings* dialog box that automatically opens, set the *Magnetization, M_y* coefficient to **1** and **-1** in the magnet ends, and **0** in the outer and magnet curved domains.

7. Select *(P2/Q2) second order conforming* in the *FEM Discretizations* drop down box to icrease the spatial discretization accuracy.

8. In *Boundary* mode, leave all the boundary conditions to the default *Magnetic insulation/antisymmetry* selection.

9. Now that the problem is specified, press the Solve mode button to switch to *solve* mode. Then press the button with an *equals* sign = to solve the problem.

10. After the problem has been solved, FEATool automatically changes to *Postprocessing* mode and plots the Magnetic potential *Vm*.



11. Contours and arrow plots of the magnetic field can be selected by using the the *Postprocessing Settings* dialog box.

12. To verify the solution, choose the *Subdomain Integration...* option in the *Post* menu, and integrate the *Magnetic potential, Az* and *Magnetic field* over all subdomains, the integrals should result in values around *-9.6e-11* and *0.0051*, respectively.

7.8.1 Magnetic Field Around a Horseshoe Magnet using the CLI

The process to set up and solve the magnetostatics problem on the command line interface is illustrated in the ex_magnetostatics2 script file which can be found in the examples directory.

7.9 Multiphysics - Natural Convection in a Square Cavity

This section contains an example how to easily define and couple multiphysics models in FEATool. The chosen example is a benchmark model by De Vahl [9], [10] which simulates natural convection in a unit square through the Boussinesq approximation. The model consists of a Navier-Stokes equations physics mode representing the fluid flow with solid wall or no-slip boundary conditions everywhere. In addition a heat transfer physics mode is added for the temperature. The top and bottom boundaries are perfectly insulated while the left boundary has a temperature of 1 and the right zero.

The physics modes are two way coupled through the vertical source term in the Navier-Stokes equations, **Pr**$*$**Ra**$*$**T** (here the Prandtl and Rayleigh numbers are, *Pr=0.71* and *Ra=1e3* respectively), and the velocities transporting the temperature coming directly from the fluid flow. The references contain benchmark reference and comparison results for a number of quantities such as maximum velocities and the Nusselt number.

7.9.1 Natural Convection in a Square Cavity using the GUI

In the following it is described how to set up and solve a multiphysics natural convection benchmark problem the FEATool graphical user interface (GUI).

1. Start Octave or Matlab, and if you have not run the installation script (which automatically adds the FEATool directory paths at startup) then change your working directory to where your FEATool installation is, for example

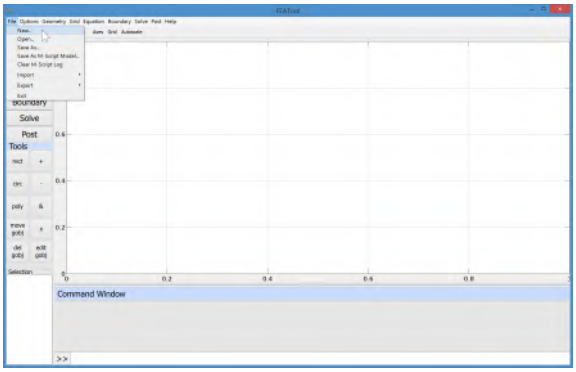
   ```
   cd C:\featool
   ```

2. In the command window type

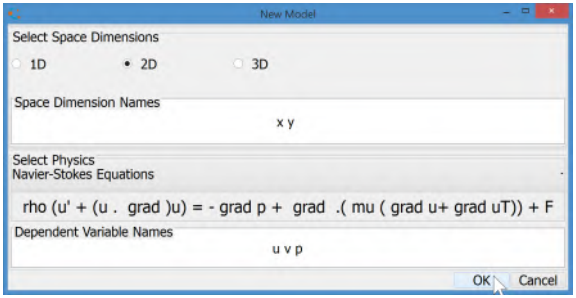
   ```
   featool
   ```

to start the graphical user interface (GUI).

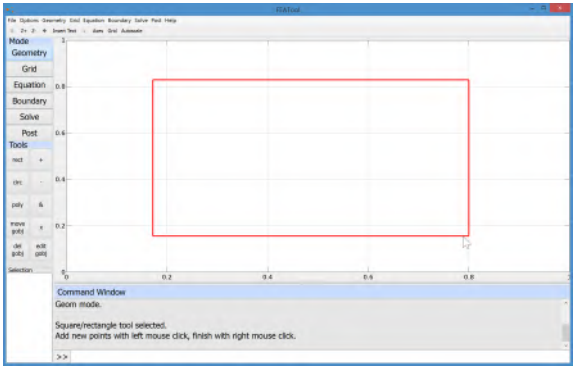
3. Either select **New...** from the **File** menu, or click on the **New Model** button in the upper horizontal toolbar, to clear all data and start defining a new model.



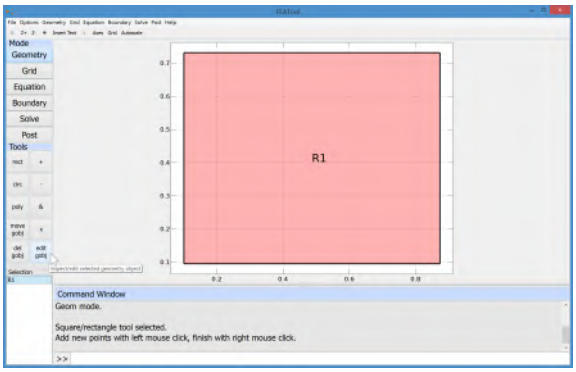
4. In the opened *New Model* dialog box, click on the **2D** radio button in the *Select Space Dimensions* frame, and select **Navier-Stokes Equations** from the *Select Physics* drop-down list. Leave the space dimension and dependent variable names to their default values. Finish and close the dialog box by clicking on the **OK** button.



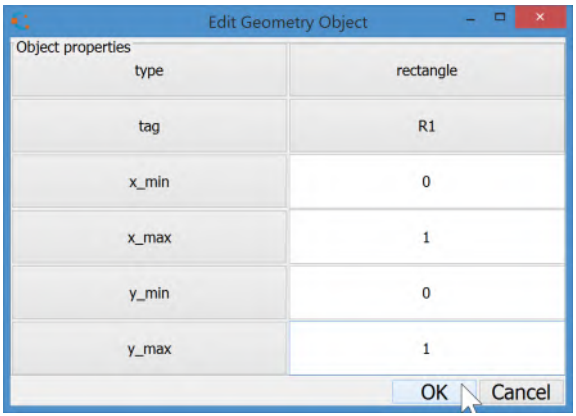
5. To create the square geometry, click on the **Create square/rectangle** button in the left hand side *Tools* toolbar frame. Then left click in the main plot axes window, hold the mouse button, and move the mouse pointer to show a rectangle with red outlines.



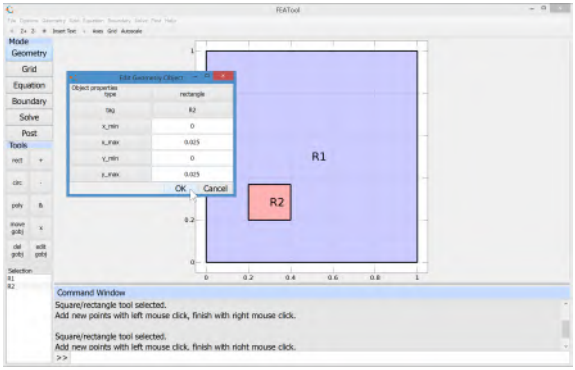
6. Release the button to finalize and create a solid geometry object. The object properties must now be edited to set the correct size and position of the rectangle. To do this, click on the rectangle *R1* to select it and highlight it in red. (Alternatively you select it by clicking on *R1* in the selection list box under the left side toolbar buttons.) Then click on the **Inspect/edit selected geometry object** toolbar button



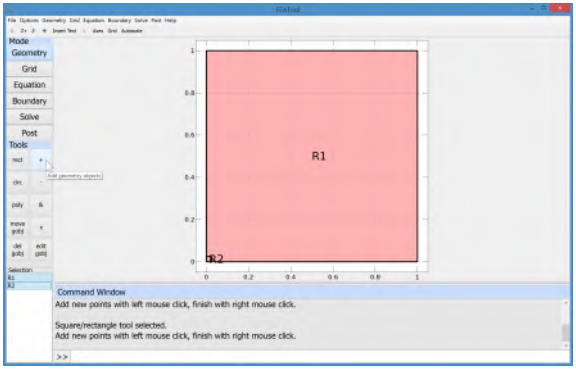
7. In the opened **Edit Geometry Object** dialog box, edit the minimum coordinates to 0 and maximum 1 to specify a unit square. Finish editing the geometry object and close the dialog box by clicking **OK**.



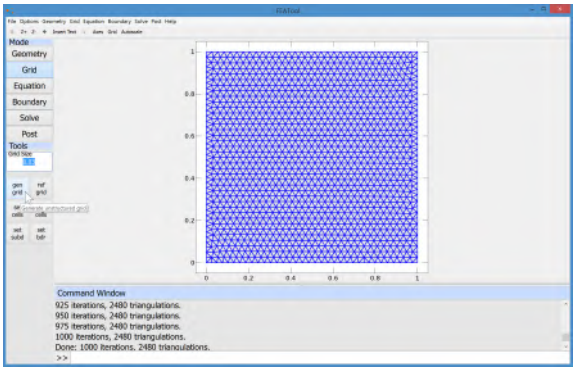
8. An artificial point must now be created to prescribe a fixed pressure since there are no natural outflow conditions in this model (this is a requirement of the Navier-Stokes equations with enclosed domains). In the following we will create small boundary segments to approximate a point. To do this add another small rectangle in the lower left corner (in this case with side length **0.0025**). Alternatively, it is also possible to use a point constraint for pressure which can be set by using **Add Point Constraints...** in the **Boundary** menu.



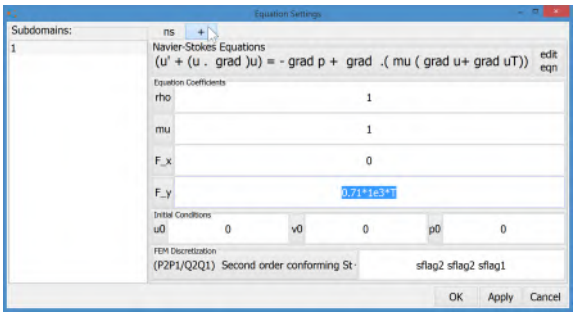
9. Join both rectangles by selecting both **R1** and **R2** and clicking on the **Add geometry objects** button .



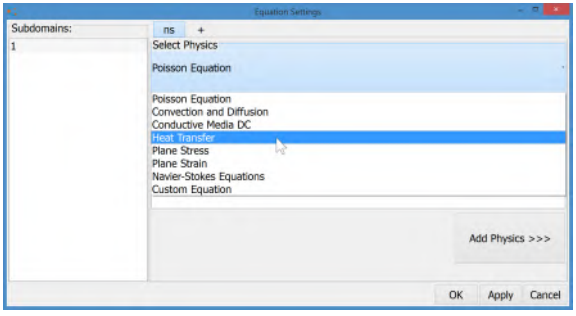
10. Switch to *Grid* mode by clicking on the corresponding *Mode* toolbar button. Then enter **0.03** in the *Grid Size* edit field and click on the **Generate** button [Generate] to call the automatic grid generation function.



11. Press the [Equation] mode button to switch from *grid* mode to *physics and equation/subdomain* specification mode. In the *Equation Settings* dialog box that automatically opens, set the fluid density ρ and viscosity μ both to **1**, and the body/volume force in the *y*-direction to **0.71∗1e3∗T** (this corresponds to the *Pa∗Ra∗T*, where T is the dependent variable for temperature which will be added next).

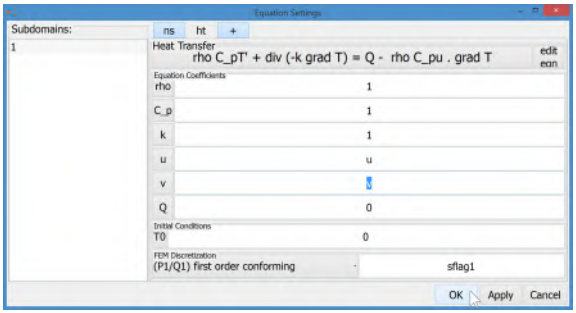


12. To access the multiphysics selection and add another physics mode press the plus + tab (only available in the Multiphysics and Professional editions of FEATool). Now select **Heat Transfer** from the *Select Physics* drop down list. Finish by pressing the **Add Physics** >>> button.

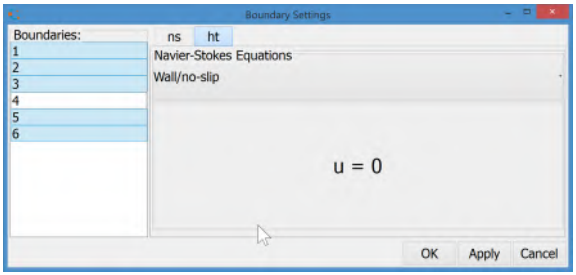


13. In the *Equation Settings ht* tab, set the density ρ, specific heat Cp, and heat conductivity k to **1**, while the source term Q is set to **0**. The convective velocities should be coupled from

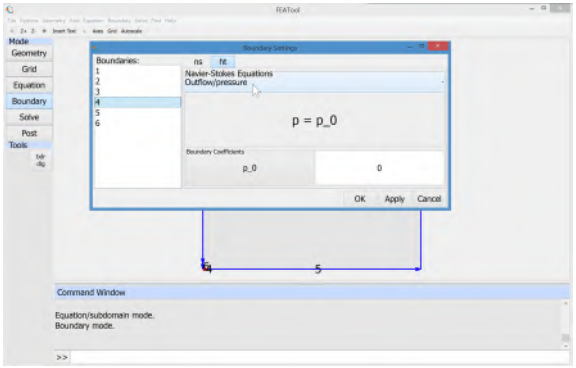
the Navier-Stokes equations physics mode, to do this enter **u** and **v** in the corresponding edit fields (as these are the default names of the dependent variables for the velocities). Press **OK** to finish with the equation specifications.



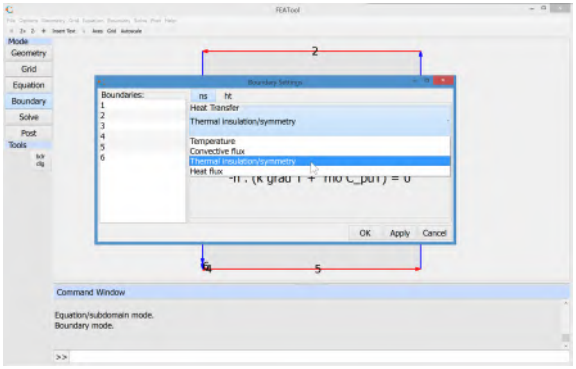
14. Switch to *boundary condition specification* mode by clicking on the Boundary mode button. Switch to the **ns** tab, which corresponds to the boundary conditions prescribed to the Navier-Stokes equations physics mode. Then select the **Wall/no-slip** for all boundaries except one of the small boundaries in the lower left corner (here all except boundary number 4).



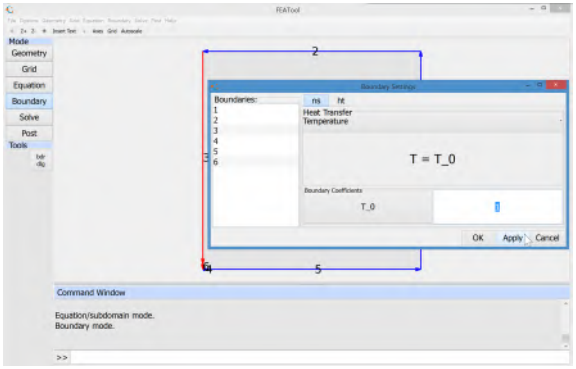
15. Select the left over boundary (number 4) and set an **Outflow/pressure** with p_o set to **0** here. This prescribes a zero fixed pressure at this particular boundary and makes the problem well posed numerically.



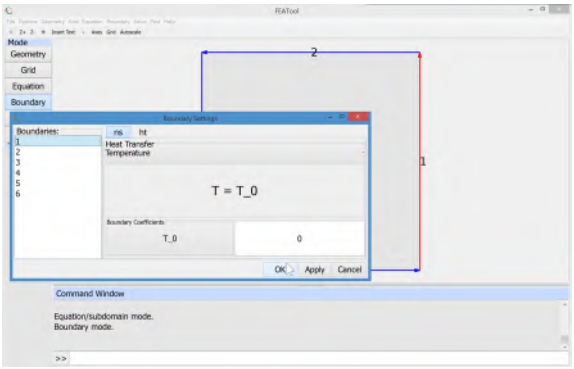
16. Click on the **ht** tab to change to specifying boundary conditions for the heat transfer physics mode. Select **Thermal insulation/symmetry** for the top and bottom boundaries (here boundary numbers 2, 4, and 5).



17. Select a **Temperature** boundary condition for the left boundary (here boundaries 3 and 6) with a fixed temperature T_o set to **1**.

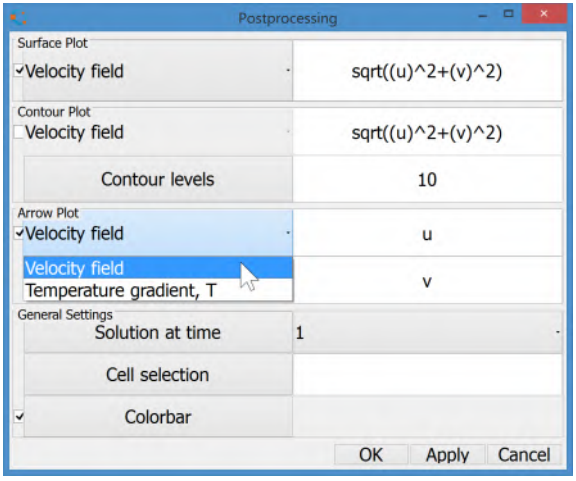


18. Similarly, also select the **Temperature** condition for the right boundary (boundary 1) but with a zero temperature, T_o set to **0**. Finish by clicking the **OK** button.

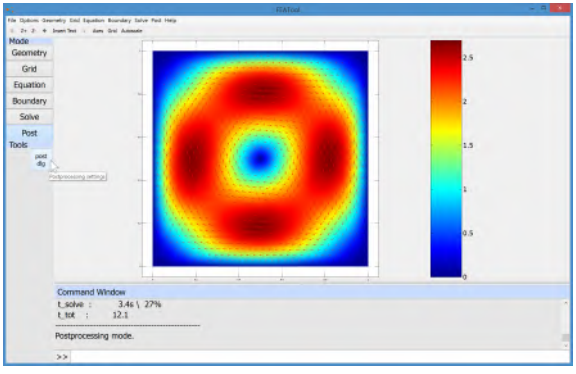


19. Now that the problem is specified, press the Solve mode button to switch to *solve* mode. Then press the button with an *equals* sign = in the *Tools* toolbar frame to call the solver with the default solver settings.

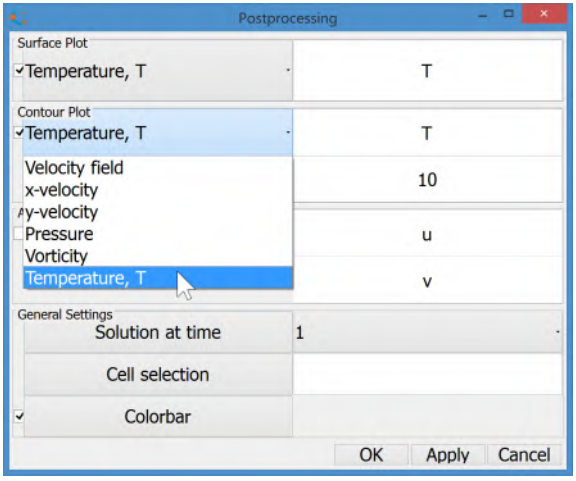
20. FEATool will automatically switch to postprocessing mode after a solution has been found. To change the plot, open the postprocessing settings dialog box by clicking on the **Post-processing settings** button. To see the velocity select **Velocity field** in the predefined *Surface Plot* and *Arrow Plot* listbox options.



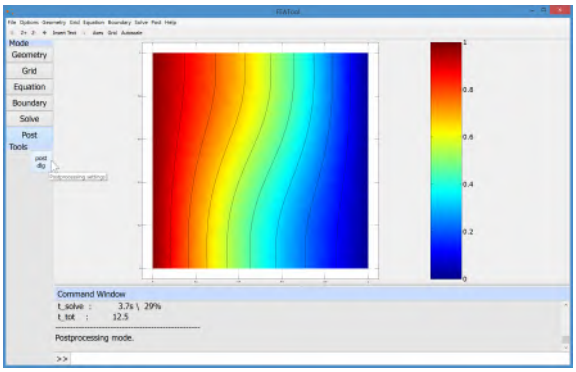
21. Press **OK** or **Apply** to show the resulting plot for the velocity field.



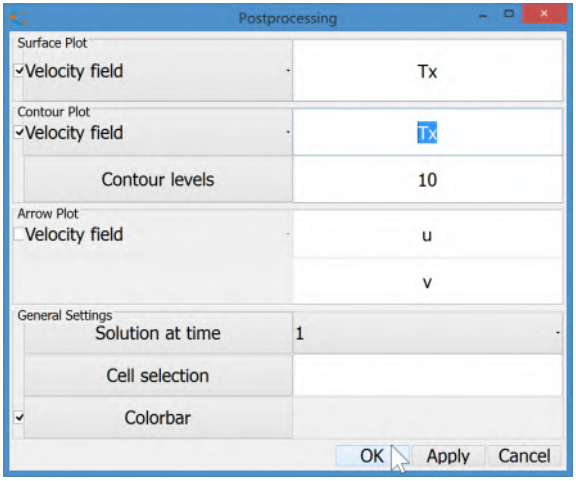
22. Similarly plot the temperature by selecting **Temperature, T** in the drop down boxes for *Surface* and *Contour* plots.

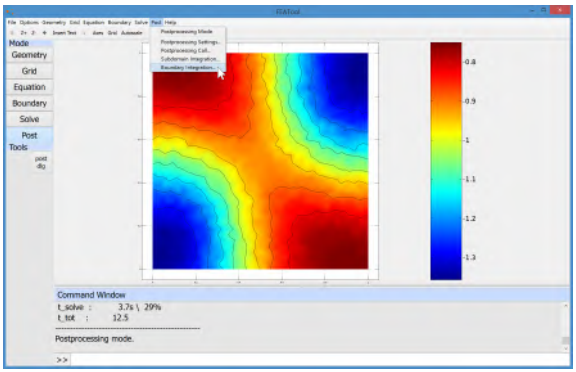


23. Press **OK** or **Apply** again to show the resulting plot for the temperature field.

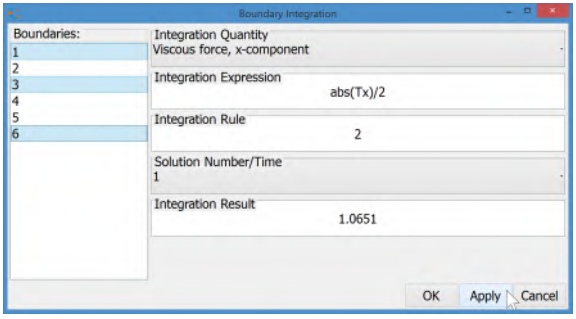


24. It is possible to enter user defined postprocessing expressions, here showing how to enter the temperature gradient in the x-direction by simply entering **Tx** in the edit fields (*T* for the dependent variable name for the temperature and *x* for the derivative in the x-direction).





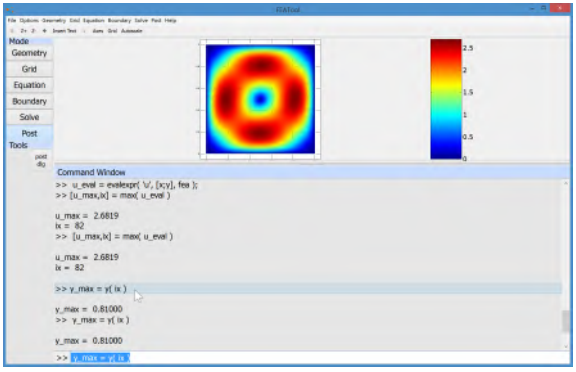
25. The professional edition of FEATool also features a boundary integration option which can be used to calculate the average Nusselt number for the vertical boundaries. Here it is showing the resulting value of 1.0651 which can be compared with the reference value of 1.118).



26. The lower command prompt can be used to input Octave and Matlab commands for advanced processing (or alternatively the whole FEATool **fea** struct can be exported to the main workspace for further processing). The following code shows how to calculate the maximum velocity and position along the vertical center line

```
y = [0:0.01:1];
x = 0.5*ones(size(y));

u_eval = evalexpr( 'u', [x;y], fea );
[u_max,ix] = max( u_eval )
y_max = y( ix )
```

The resulting values of *u_max* and *y_max* are here 2.6819 and 0.81, respectively, which is a little low compared to the reference solution (3.649 at a height of 0.813) which can indicate the the grid resolution is somewhat too coarse.



7.9.2 Natural Convection in a Square Cavity using the CLI

The process to set up and solve the natural convection problem on the command line interface is illustrated in the ex_natural_convection script file which can be found in the examples directory.

7.9.3 References

[9] D. de Vahl Davis, Natural Convection of Air in a Square Cavity: A Bench Mark Solution, Int. J. Numer. Meth. Fluids, vol. 3, pp. 249-264, 1983.

[10] D. de Vahl Davis and I. P. Jones, Natural Convection of Air in a Square Cavity: A Comparison Exercise, Int. J. Numer. Meth. Fluids, vol. 3, pp. 227-248, 1983.

7.10 Multiphysics - Heat Exchanger

An example of setting up and modeling a heat exchanger with coupled fluid flow and temperature with both free and forced convection is described in the FEATool quickstart guide Multiphysics Example - Heat Exchanger.

7.11 Multiphysics - Piezoelectric Bending of a Beam

This example describes how to model piezoelectric bending of a beam on the command line as an m-script model in FEATool. The test case comes from a research paper by Hwang and Park [11]. For this problem, the following coupled system of equations should be solved

$$-\nabla \cdot \begin{bmatrix} \sigma \\ D \end{bmatrix} = \begin{bmatrix} f \\ -\rho \end{bmatrix}$$

where σ is a stress tensor, f represents body forces, D electric displacement, and ρ distributed charges.

By using the constitutive relations for a piezoelectric material [11], the stresses can in two dimensions be converted to the following form

$$\nabla \cdot \left[C \begin{bmatrix} \nabla u \\ \nabla v \\ \nabla V \end{bmatrix} \right] = \begin{bmatrix} 0 \\ 0 \\ -\rho \end{bmatrix}$$

where C is an array of constitutive relations and material parameters. This leads to a system of three equations for the unknown displacements u and v, and the potential V.

There are no readily available physics modes in FEATool yet so to model this we are defining the problem from scratch. In this case we will be looking at composition of two 1.2 cm long and 2 mm thick beams with opposite polarization.

First we assign names to the space coordinates (x and y), and create a grid. Note, that the shape is so simple here that we don't need to explicitly define a geometry, we can use the 'rectgrid' command directly

```
fea.sdim = { 'x' 'y' };
fea.grid = rectgrid( 20, 20, [ 0 12e-3; -1e-3 1e-3] );
```

Next is to define the equation coefficients and constitutive relations. See the attached file for their definitions, one thing to note that to define the polarization the coefficient is multiplied with a switch expression **2*(y<0)-1** which evaluates to 1 in the top half and -1 in the bottom (here we have also used the name y for the space coordinate in the 2nd direction as we defined earlier).

Now we define the dependent variables and assign 2nd order Lagrange finite element shape functions for them all

```
fea.dvar  = { 'u' 'v' 'V' };
fea.sfun  = { 'sflag2' 'sflag2' 'sflag2' };
```

The material parameters and constitutive relations are defined as

```
% Equation coefficients.
Emod =  2e9;       % Modulus of elasticity
nu   =  0.29;      % Poissons ratio
Gmod =  0.775e9;   % Shear modulus
d31  =  0.22e-10;  % Piezoelectric sn coefficient
d33  = -0.3e-10;   % Piezoelectric sn coefficient
prel = 12;         % Relative electrical permittivity
pvac = 0.885418781762-11; % Elec. perm. of vacuum

% Constitutive relations.
constrel   = [ Emod/(1-nu^2)    nu*Emod/(1-nu^2) 0    ;
               nu*Emod/(1-nu^2) Emod/(1-nu^2)    0    ;
               0                0                Gmod ];
piezoel_st = [ 0 d31 ;
               0 d33 ;
               0 0   ];
piezoel_sn = constrel*piezoel_st;
dielmat_st = [ prel 0    ;
               0    prel ]*pvac ;
dielmat_sn = [ dielmat_st - piezoel_st'*piezoel_sn ];

% Populate coefficient matrices (negative sign
% due to fem partial integration).
c{1,1} = { constrel(1,1)   constrel(1,3) ;
           constrel(1,3)   constrel(3,3) };
c{1,2} = { constrel(1,3)   constrel(1,2) ;
           constrel(3,3)   constrel(2,3) };
c{2,1} = c{1,2}';
c{2,2} = { constrel(3,3)   constrel(1,3) ;
           constrel(1,3)   constrel(2,2) };
c{1,3} = { piezoel_sn(1,1) piezoel_sn(1,2) ;
```

```
            piezoel_sn(3,1) piezoel_sn(3,2) };
for i=1:4
  c{1,3}{i} = [num2str(c{1,3}{i}),'*(2*(y<0)-1)'];
end
c{3,1} = c{1,3}';
c{2,3} = { piezoel_sn(3,1) piezoel_sn(3,2);
            piezoel_sn(2,1) piezoel_sn(2,2) };
for i=1:4
  c{2,3}{i} = [num2str(c{2,3}{i}),'*(2*(y<0)-1)'];
end
c{3,2} = c{2,3}';
c{3,3} = { dielmat_sn(1,1) dielmat_sn(2,1) ;
            dielmat_sn(2,1) dielmat_sn(2,2) };
```

To define the finite element bilinear forms we use the following code

```
bilinear_form = [ 2 2 3 3 ;
                  2 3 2 3 ];
for i=1:length(fea.dvar)
  for j=1:length(fea.dvar)
    fea.eqn.a.form{i,j} = bilinear_form;
    fea.eqn.a.coef{i,j} = c{i,j}(:)';
  end
end
```

where each bilinear form has four terms, the top line in the definition defines the finite element shape for the dependent variable, and second line for the test function space (a 2 indicates x-derivative, and 3 y-derivative).

In this case all three source terms are zero, thus

```
fea.eqn.f.form = { 1 1 1 };
fea.eqn.f.coef = { 0 0 0 };
```

The boundary conditions are defined as follows

```
n_bdr                   = max(fea.grid.b(3,:));
```

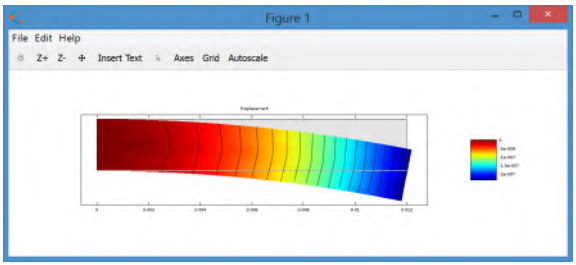
```
fea.bdr.d                = cell(3,n_bdr);
fea.bdr.n                = cell(3,n_bdr);
i_top                    = 3;
i_bottom                 = 1;
i_left                   = 4;
[fea.bdr.d{3,i_top}]     = deal( 200 );
[fea.bdr.d{3,i_bottom}] = deal( 0 );
[fea.bdr.d{1:2,i_left}] = deal( 0 );
```

In this way we have set a potential difference of 200, and fixed the left hand side boundary. Now we parse the problem struct and solve the system with

```
fea       = parseprob( fea );
fea.sol.u = solvestat( fea );
```

After the problem has been solved we can for example visualize the displacement in the y-direction with the following commands

```
plotsubd( fea, 'labels', 'off', 'setaxes', 'off' )
u = evalexpr( 'u', fea.grid.p, fea );
v = evalexpr( 'v', fea.grid.p, fea );

DSCALE = 5000;
fea_disp = fea;
fea_disp.grid.p = fea.grid.p + [ u v ]'*DSCALE;
postplot( fea_disp, 'surfexpr', 'v', 'isoexpr', 'v' )
```



An example m-script file for this model can be found in the ex_piezoelectric1 m-script file.

7.11.1 References

[11] Hwang, W. S., Park, H. C. Finite Element Modeling of Piezoelectric Sensors and Actuators. AIAA Journal, 31(5), pp 930-937, 1993.

7.12 Custom Equation - Black-Scholes model equation

This section contains an example how to easily define a custom equation or model in FEATool. In this case a Black-Scholes model equation is studied which is often used in financial analytics. The equation to be modeled reads

$$\frac{\partial u}{\partial t} - \frac{\partial}{\partial x}(\frac{1}{2}\frac{\partial u}{\partial x}) - \frac{\partial u}{\partial x} = -u + ((x-t)^5 - 10(x-t)^4 - 10(x-t)^3)$$

with boundary conditions $u(0,t) = -t^5$ and $u(x,t) = (x-t)^5$ on the left and right sides of the domain, respectively. Initial condition is taken as $u(x,t=0) = x^5$ For this problem an exact analytical solution exists, namely $u(x,t) = (x-t)^5$. Consult the Custom Equation section for details on the equation syntax.

7.12.1 Black-Scholes model equation using the GUI

This section describes how to set up and solve the custom equation example with the FEATool graphical user interface (GUI).

1. Start Octave or Matlab, and if you have not run the installation script (which automatically adds the FEATool directory paths at startup) then change your working directory to where your FEATool installation is, for example

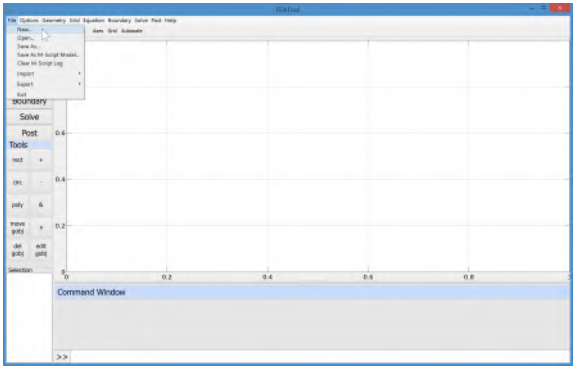
   ```
   cd C:\featool
   ```

2. In the command window type

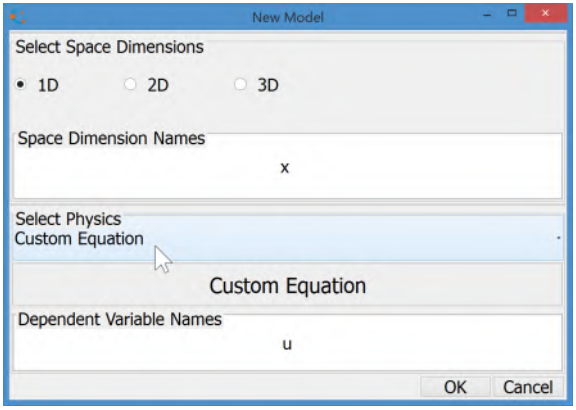
   ```
   featool
   ```

   to start the graphical user interface (GUI).

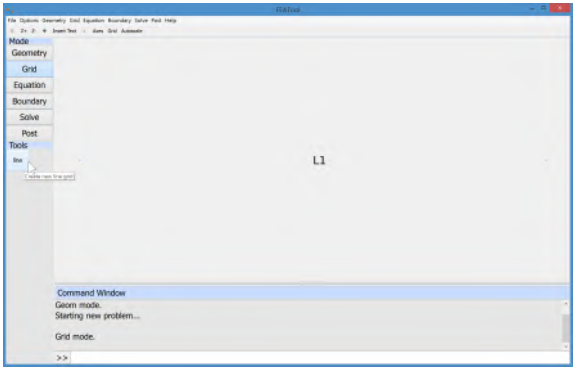
3. Click on the **New Model** button in the upper horizontal toolbar to clear all data and start defining a new model.



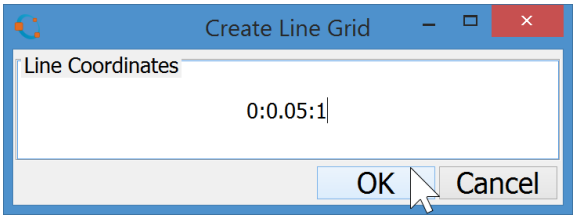
4. In the opened *New Model* dialog box, click on the **1D** radio button in the *Select Space Dimensions* frame, and select **Custom Equation** from the *Select Physics* drop-down list. Leave the space dimension and dependent variable names to their default values. Finish and close the dialog box by clicking on the **OK** button.



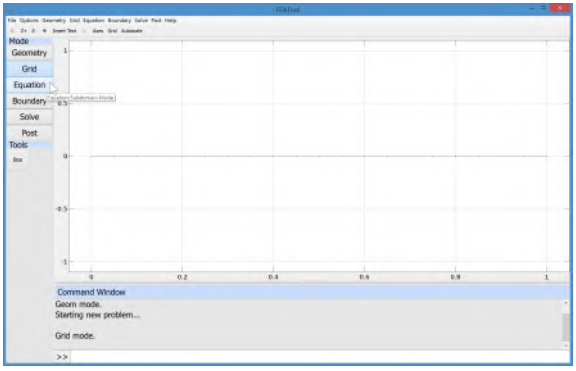
5. To create a line grid, first click on the **Create new line grid** button in the left hand side *Tools* toolbar frame.



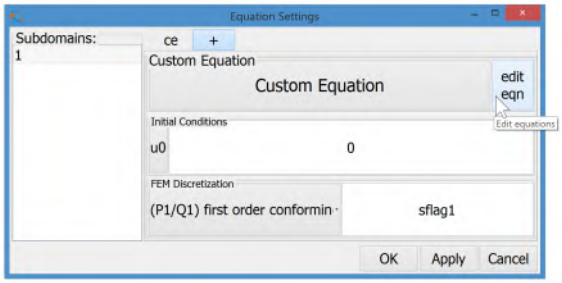
6. Enter **0:0.05:1** in the *Line Coordinates* edit field of the *Create Line Grid dialog* box. This will specify a grid between *0* and *1* with *20 (1/0.05)* cells. Press **OK** to finish and close the dialog box.



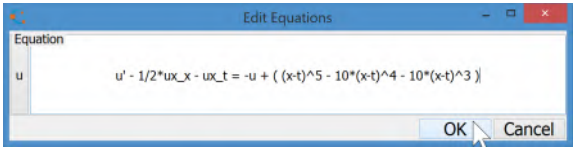
7. Press the `Equation` mode button in the *Mode* toolbar to switch from *grid* mode to *physics and equation/subdomain* specification mode.



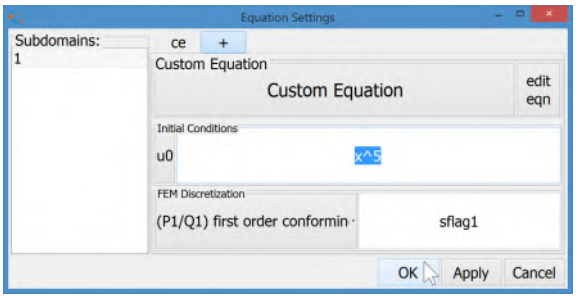
8. Press the **edit eqn** button in the *Equation Settings* dialog box that automatically opens.



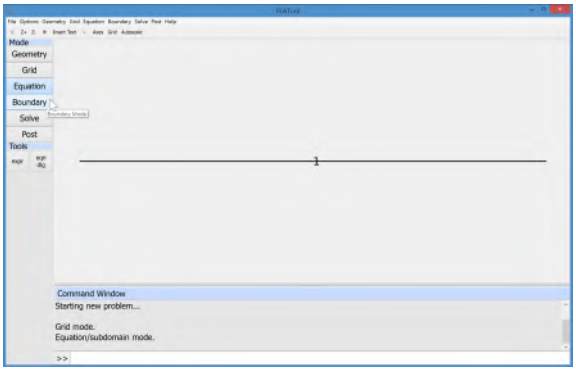
9. Enter **u' - 1/2∗ux_x - ux_t = -u + ((x-t)^5 - 10∗(x-t)^4 - 10∗(x-t)^3)** in the *Edit Equations* dialog box. Press **OK** to finish.



10. Enter **x^5** in the *Initial Conditions* edit field. Press **OK** to finish with the equation specifications and close the dialog box.

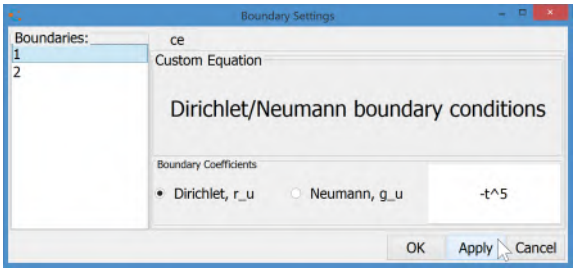


11. Switch to *boundary condition specification* mode by clicking on the `Boundary` mode button

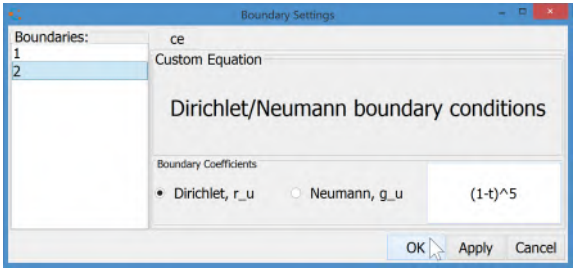


12. In the *Boundary Settings* dialog box, first select the left boundary, boundary **1** in the *Boundaries:* list box. Choose the *Dirichlet, r_u* condition, and enter -t^5 in the corresponding edit

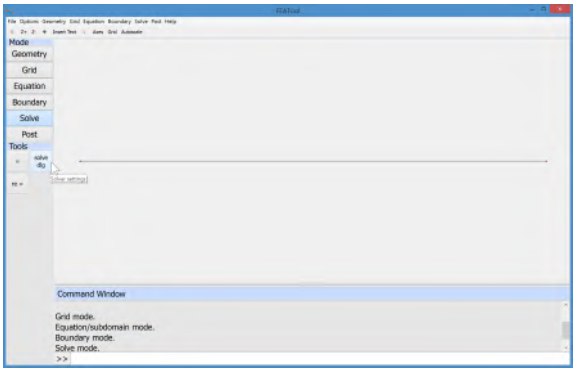
field.



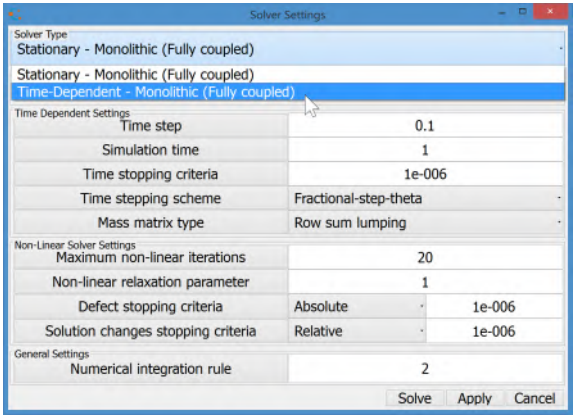
1. Similarly for the right boundary, select boundary **2** in the *Boundaries:* list box and choose the *Dirichlet, r_u* condition, and enter (1-t)^5 in the corresponding edit field. Press **OK** to finish the boundary condition specifications.



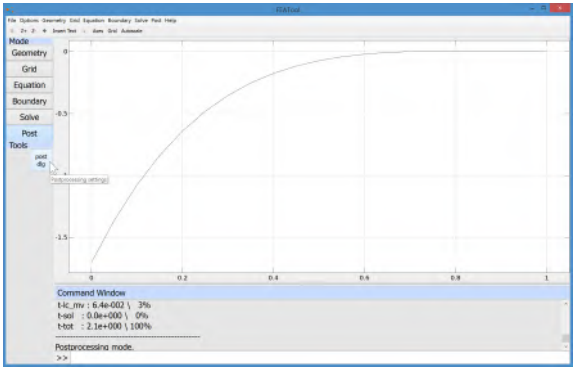
2. Now that the problem is specified, press the Solve mode button to switch to *solve* mode. Then press the **Solver settings** toolbar button.



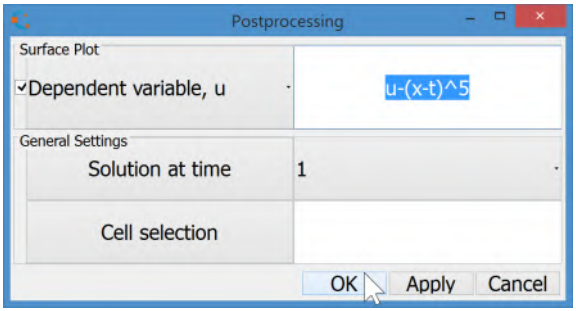
3. Select the **Time-Dependent - Monolithic (Fully coupled)** solver option, then press **Apply** and **Solve** to start the solution process.



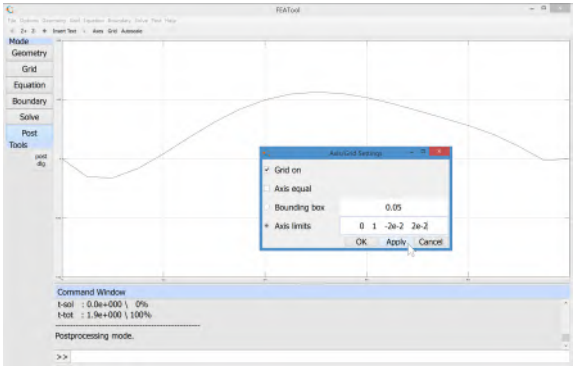
4. After the problem has been solved FEATool will automatically switch to *postprocessing* mode and display the computed solution at the last time step. To change the plot open the postprocessing settings dialog box by clicking on the **Postprocessing settings** button in the *Tools* toolbar frame.



5. In the Postprocessing settings dialog box enter **u-(x-t)^5** to show the difference between the computed and exact analytical solution.



6. To zoom in on the error open the **Axis/Grid Settings...** in the **Options** menu. Switch off **Axis equal** and manually set the **Axis limits** to *0 1 -2e-2 2e-2*. We can now see that the maximum error has a magnitude of *1e-2* which can be further improved by using a finer grid and smaller time steps.



7.12.2 Black-Scholes model equation using the CLI

The process to set up and solve the custom equation problem on the command line interface is illustrated in the ex_custom_equation1 script file which can be found in the examples directory.

7.13 Custom Equation - Wave Equation

In the quickstart guide there is described how to set up a model of the wave equation with the custom equations physics mode (Custom Equation Example - Wave Equation on a Circle). In this case the custom equation uses two equations/dependent variables do describe the system.

7.14 Classic Equation - Poisson Equation on a Circle

The classic Poisson equation is one of the most fundamental partial differential equations (PDEs). Although one of the simplest equations, it is a very good model for the process of diffusion and comes up again and again in many applications (for example fluid flow, heat transfer, and chemical transport). It is thus very fundamental to many simulation codes to be able to solve it correctly and efficiently.

This example shows how to up and solve the Poisson equation

$$d_{ts}\frac{\partial u}{\partial t} + \nabla \cdot (-D\nabla u) = f \tag{1}$$

for a scalar field $u = u(\mathbf{x})$ on a circle Ω with radius $r = 1$ in two dimensions. Both the diffusion coefficient D and right hand side source term f are assumed constant and equal to 1. The Poisson problem is also considered stationary meaning the time dependent term can be neglected. With these assumptions equation (1) simplifies to

$$-\Delta u = 1.$$

Moreover, homogeneous Dirichlet boundary conditions are prescribed on all boundaries of the domain, that is $u = 0\ on\ \partial\Omega$. The exact solution for this problem is $u(x,y) = (1 - x^2 - y^2)/4$ which can be used to measure the accuracy of the computed solution.

7.14.1 Poisson Equation using the GUI

This section describes how to set up and solve the Poisson equation (1) with the FEATool graphical user interface (GUI) which is available when using FEATool together with Octave version 4.0 or later and Matlab.

1. Start Octave/Matlab, and if you have not run the installation script (which automatically adds the FEATool directory paths at startup) then change your working directory to where your FEATool installation is, for example

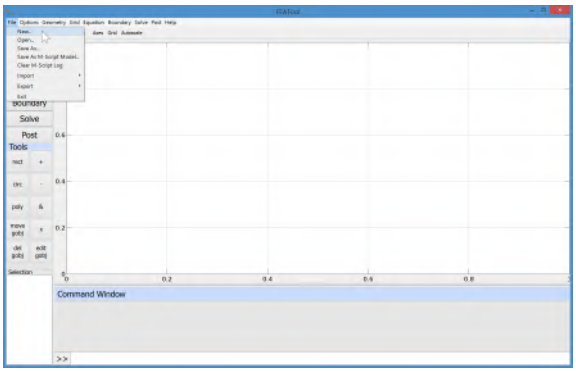
```
cd C:\featool
```

2. In the command window type

```
featool
```

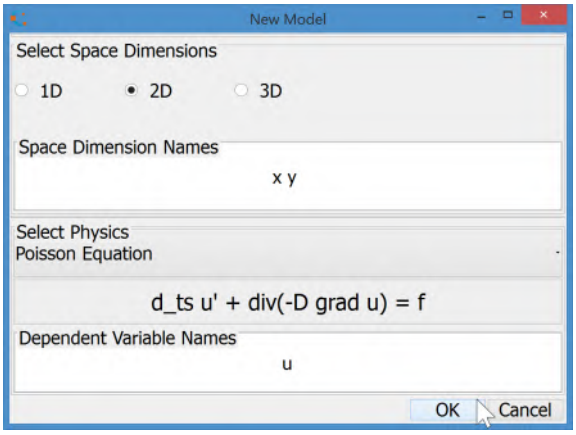
   to start the graphical user interface (GUI).

3. Either select **New...** from the **File** menu, or click on the **New Model** button in the upper horizontal toolbar, to clear all data and start defining a new model.

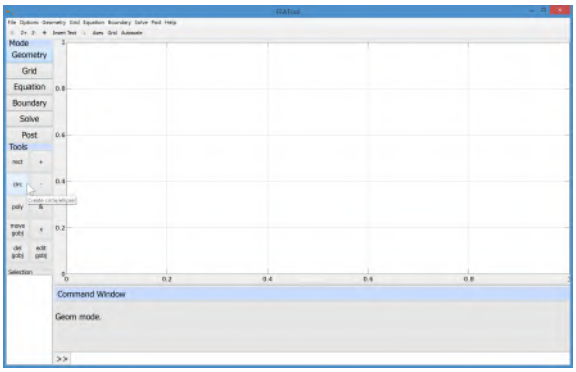


4. In the opened *New Model* dialog box, click on the **2D** radio button in the *Select Space Dimensions* frame, and select **Poisson Equation** from the *Select Physics* drop-down list.

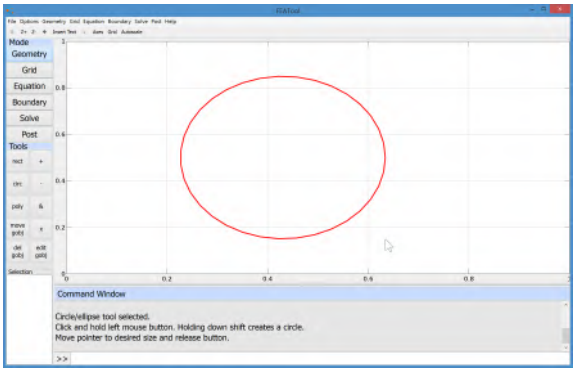
Leave the space dimension and dependent variable names to their default values. Finish and close the dialog box by clicking on the **OK** button.



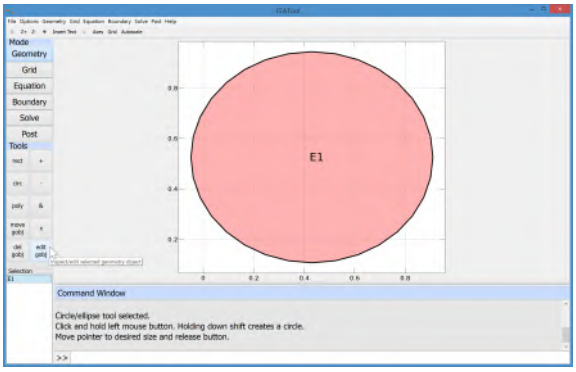
5. To create a circle, first left click on the **Create circle/ellipse** button in the left hand side *Tools* toolbar frame.



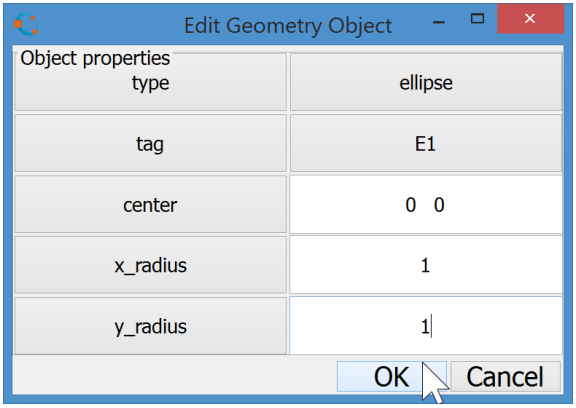
6. Then click and hold the left mouse button anywhere in the main plot axes window, and move the mouse pointer to show red outlines of a circle or ellipse. Release the button to finalize and create a solid geometry object.



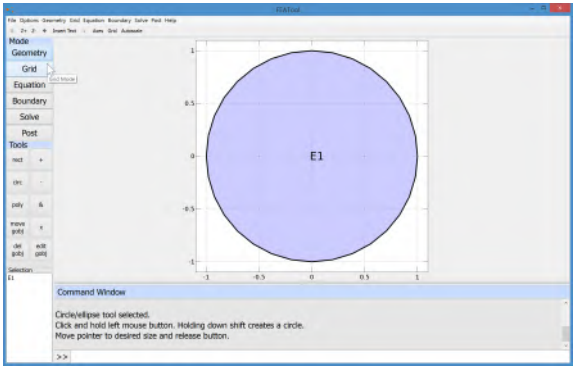
7. The object properties must be changed to make a circle with radius 1 centered at the origin. To do this, click on the ellipse *E1* to select it which also highlights it in red (alternatively you select it by clicking on *R1* in the selection list box under the left side toolbar buttons). Then click on the **Inspect/edit selected geometry object** toolbar button



8. In the opened **Geometry Object** dialog box change the *center* coordinates edit field **0 0**, and the *x radius* and *y radius* to **1** in the corresponding fields. Finish editing the geometry object and close the dialog box by clicking **OK**.

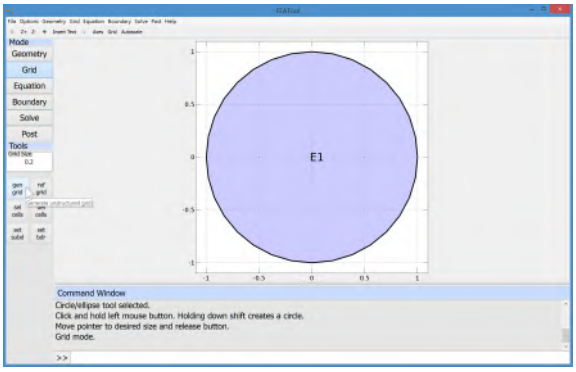


9. Press the Grid mode button in the *Mode* toolbar to switch from *geometry* mode to *grid* generation mode.



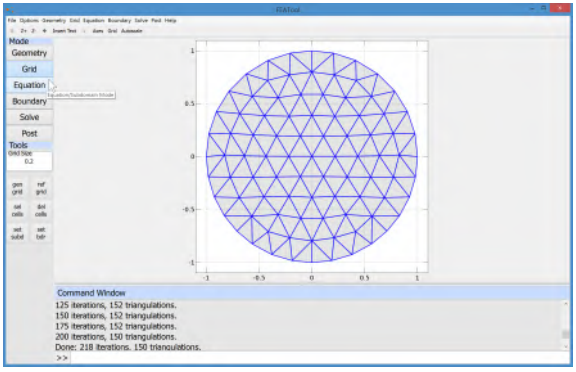
10. Click on the **Generate** button

to call the grid generation function which automatically generates a grid of triangles for the circle.

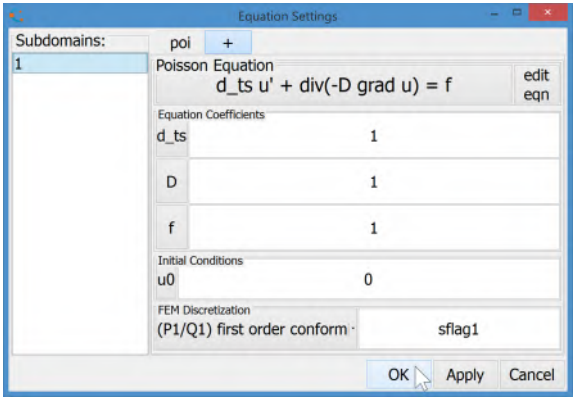


11. Press the

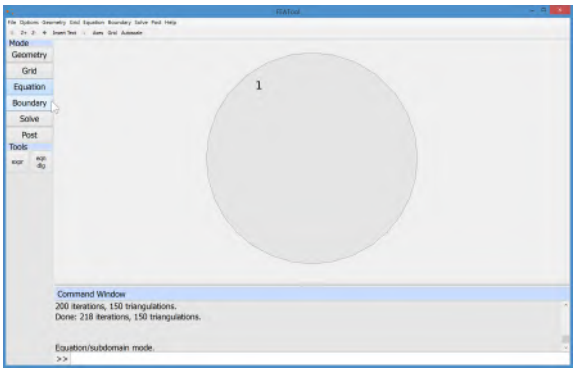
mode button in the *Mode* toolbar to switch from *grid* mode to *physics and equation/subdomain* specification mode.



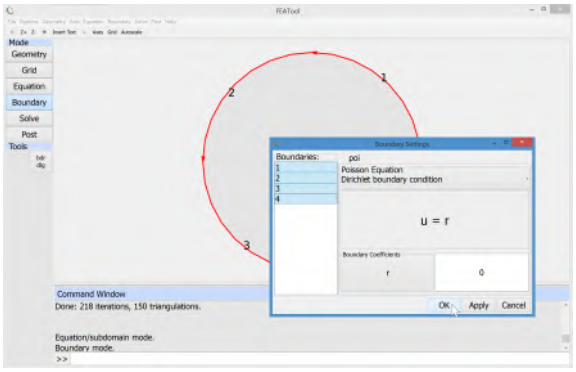
12. In the *Equation Settings* dialog box that automatically opens, set both the diffusion coefficient D and source term coefficient f to **1** in the corresponding edit fields. All other coefficients can be left to their default values. Press **OK** to finish and close the dialog box.



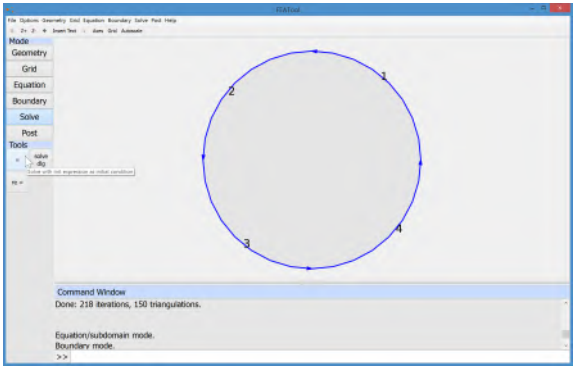
13. Switch to *boundary condition specification* mode by clicking on the Boundary mode button



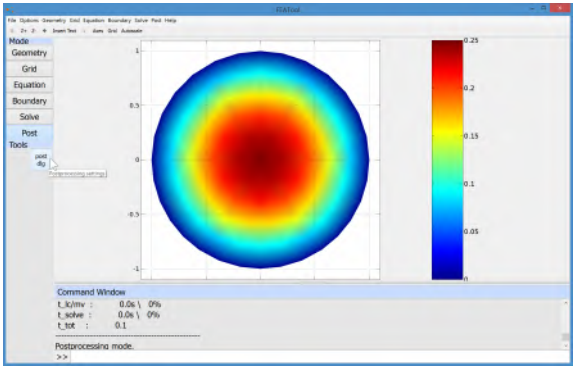
14. In the *Boundary Settings* dialog box, select all boundaries in the left hand side *Boundaries* list box and choose **Dirichlet boundary condition** in the drop-down list. Set the Dirichlet boundary coefficient ***r*** equal to **0** in the *Boundary Coefficients* frame and finish by clicking on **OK**.



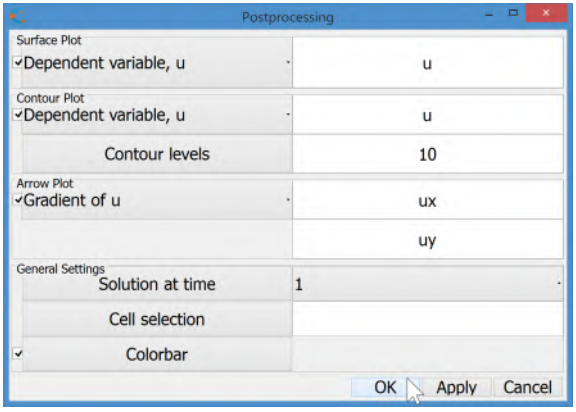
15. Now that the problem is specified, press the Solve mode button to switch to *solve* mode. Then press the button with an *equals* sign = in the *Tools* toolbar frame to call the solver with the default solver settings.



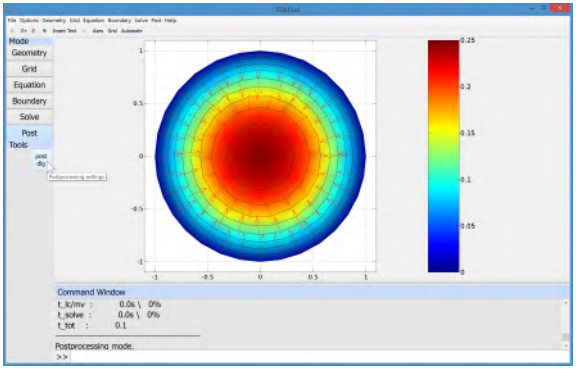
16. After the problem has been solved FEATool will automatically switch to *postprocessing* mode and display the computed solution. To change the plot, open the postprocessing settings dialog box by clicking on the **Postprocessing settings** button in the *Tools* toolbar frame.



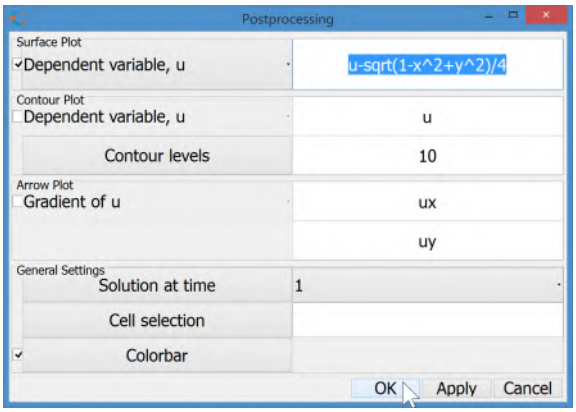
17. Activate *Contour* and *Arrow* plots by marking the corresponding check boxes and press **OK** or **Apply** to show the new plot.



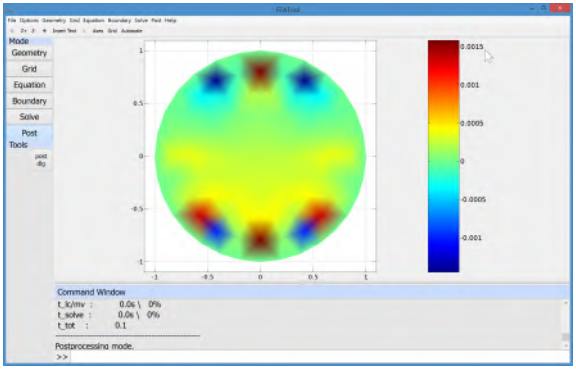
18. The solution is now visualized as surface, contour, and arrow plots in the main plot axes window.



19. By going back to the *Postprocessing* settings dialog box and entering the expression *u-(1-x^2-y^2)/4* in the *Surface Plot* expression edit field it is possible to plot and visualize the difference between the computed and exact reference solution.



20. The error in the computed solution is now plotted, which clearly shows that the errors are localized to the top and bottom boundaries. From the colorbar we can also see that the maximum error has a magnitude of 1.5e-3.



21. To further reduce the error one can recompute the problem with a finer grid, and also increase the solution space approximation order by choosing a higher finite element shape function in the subdomain settings.

7.14.2 Poisson Equation using the CLI and Physics Modes

This section describes how to set up and solve the Poisson equation (1) with the Octave or Matlab command line interface (CLI). To make it easier and faster for users to set up model problems several physics modes have been predefined with equations and boundary conditions for physics such as convection, diffusion, and reaction of chemical species, heat transport through convection and conduction, and incompressible fluid flow with the Navier-Stokes equations. In this example the Poisson physics mode will be used.

1. Start Octave/Matlab and make sure you have either run the installation script to set up the FEATool directory paths or alternatively run

   ```
   addpaths
   ```

   on the command line from the FEATool installation directory before you start.

2. Define a circle geometry object

   ```
   r    = 1;     % Circle radius.
   xc   = 0;     % x-center coordinate.
   yc   = 0;     % y-center coordinate.

   gobj = gobj_circle( [xc yc], r );
   fea.geom.objects  = { gobj };
   ```

3. Call gridgen to create an unstructured grid with maximum grid cell size *hmax*

```
hmax     = 0.1;   % Maximum grid cell size.
fea.grid = gridgen( fea, 'hmax', hmax );
```

4. Assign names to the space dimension coordinates (these names can then be used in equation coefficients and postprocessing expressions)

```
fea.sdim = { 'x' 'y' };
```

5. Use addphys to add the predefined Poisson physics mode poisson

```
fea = addphys( fea, @poisson );
```

6. To find and set the diffusion coefficient and source term to 1 first Inspect the 'eqn.coef' field and verify that the *d_poi* coefficient and *f_poi* source term are assigned rows 2 and 3, respectively. The coefficients values are specified in the fourth column of *eqn.coef* as a cell array of values per subdomain (in this case only one subdomain)

```
fea.phys.poi.eqn
```

7. Now set the correct coefficients

```
fea.phys.poi.eqn.coef{2,4} = {1};   % Diffusion coefficient.
fea.phys.poi.eqn.coef{3,4} = {1};   % Source term.
```

8. To find and set the Dirichlet boundary coefficients to 0 first inspect the *bdr.coef* field and verify that the Dirichlet conditions are used with the coefficient set to zero

```
fea.phys.poi.bdr.coef
```

9. Then set the correct boundary coefficients

```
% Use first boundary condition specification.
fea.phys.poi.bdr.sel       = 1;

% Set Dirichlet coefficient to zero.
fea.phys.poi.bdr.coef{1,7} = {0};

% Set flag to use Dirichlet BC for first selection.
fea.phys.poi.bdr.coef{1,5} = {1};

% Set flag not to use Neumann BC for first selection.
fea.phys.poi.bdr.coef{2,5} = {0};
```

10. Check and parse the physics and problem structs by calling parsephys and parseprob (this verifies that everything in the problem definition struct is correct and outputs the necessary lower hierarchy fields)

```
fea = parsephys( fea );
fea = parseprob( fea );
```

11. Call the stationary solver solvestat with the default settings

```
fea.sol.u = solvestat(fea);
```

12. Plot with postplot to display the scalar solution as surface and contour plots, and the solution gradient (*ux*, *uy*) as arrow plots

```
postplot( fea, 'surfexpr', 'u', ...
               'isoexpr',  'u', ...
               'arrowexpr', {'ux','uy'} )
```

13. Plot the difference between the computed and exact solution

```
u_diff = 'u-(1-x^2-y^2)/4';
clf
postplot( fea, 'surfexpr', u_diff )
```

7.14.3 Poisson Equation using the CLI without Physics Modes

This section describes how to set up and solve the Poisson equation (1) with the Octave or Matlab command line interface (CLI) by directly setting up the necessary equation and boundary definition fields (that is without using the predefined Poisson physics mode).

1. Start Octave/Matlab and make sure you have either run the installation script to set up the FEATool directory paths or alternatively run

   ```
   addpaths
   ```

   on the command line from the FEATool installation directory before you start.

2. Define a circle geometry object

   ```
   r    = 1;     % Circle radius.
   xc   = 0;     % x-center coordinate.
   yc   = 0;     % y-center coordinate.

   gobj = gobj_circle( [xc yc], r );
   fea.geom.objects  = { gobj };
   ```

3. Call gridgen to create an unstructured grid with maximum grid cell size *hmax*

```
hmax     = 0.1;   % Maximum grid cell size.
fea.grid = gridgen( fea, 'hmax', hmax );
```

4. Assign names to the space dimension coordinates (these names can then be used in equation coefficients and postprocessing expressions)

```
fea.sdim = { 'x' 'y' };
```

5. Assign a name to the dependent variable (in this case *u*)

```
fea.dvar = { 'u' };
```

6. Define a finite element shape function to use for the discretization (here sflag1 corresponds to linear conforming P1 finite element shape functions, while sflag2 would correspond to second order P2 shape functions.)

```
fea.sfun = { 'sflag1' };
```

7. Define the equation system, bilinear form in *fea.eqn.a* and right hand side linear form in *fea.eqn.f*

```
% Define bilinear form. The first row indicates test function
% space, and second row trial function space. Each column
% defines a term in the bilinear form and 1 corresponds to
% a basis function evaluation, 2 = x-derivative,
% 3 = y-derivative, (d 4 = z-derivative in 3D).
fea.eqn.a.form = { [2 3; 2 3] };

% Define coefficients used in assembling the bilinear forms.
fea.eqn.a.coef = { [1 1] };

% Test function space to evaluate in linear form.
```

```
fea.eqn.f.form = { 1 };

% Coefficient value used in assembling the linear form.
fea.eqn.f.coef = { 1 };
```

8. Define the boundary conditions, Dirichlet conditions in *fea.bdr.d* and Neumann (flux conditions) in *fea.bdr.n*

```
% Number of boundary segments.
n_bdr          = max(fea.grid.b(3,:));

% Allocate space for n_bdr boundaries.
fea.bdr.d      = cell(1,n_bdr);

% Assign u=0 to all Dirichlet boundaries.
[fea.bdr.d{:}] = deal(0);

% No Neumann boundaries (fea.bdr.n{i} empty).
fea.bdr.n      = cell(1,n_bdr);
```

9. Check and parse the problem struct by calling parseprob (this verifies that everything in the problem definition struct is correct and outputs the necessary lower hierarchy fields)

```
fea = parseprob( fea );
```

10. Call the stationary solver solvestat with the default settings

```
fea.sol.u = solvestat( fea );
```

11. Plot the results with postplot to display the scalar solution as and contour plots, and the solution gradient (*ux*, *uy*) as arrow plots

```
postplot( fea, 'surfexpr', 'u', ...
```

```
                'isoexpr',  'u', ...
                'arrowexpr', {'ux','uy'} )
```

12. Plot the difference between the computed and exact solution

```
u_diff = 'u-(1-x^2-y^2)/4';
clf
postplot( fea, 'surfexpr', u_diff )
```

7.14.4 Poisson Equation using the CLI and core FEM library functions

This final section shows how to define and solve the Poisson equation (1) by directly calling the core finite element library functions. The low level functions are used to assemble the system matrix and right hand side vector to which boundary conditions are applied, after which the system can be solved.

1. Start Octave/Matlab and make sure you have either run the installation script to set up the FEATool directory paths or alternatively run

   ```
   addpaths
   ```

   on the command line from the FEATool installation directory before you start.

2. Define a circle geometry object

   ```
   r    = 1;     % Circle radius.
   xc   = 0;     % x-center coordinate.
   yc   = 0;     % y-center coordinate.
   ```

```
gobj = gobj_circle( [xc yc], r );
fea.geom.objects  = { gobj };
```

3. Call gridgen to create an unstructured grid with maximum grid cell size *hmax*

```
hmax     = 0.1;   % Maximum grid cell size.
fea.grid = gridgen( fea, 'hmax', hmax );
```

4. Use the function *assemblea* to assemble the finite element stiffness matrix

```
% Bilinear form to assemble (u_x*v_x+u_y*v_y).
form  = [2 3;2 3];

% FEM shape functions used in test and trial function spaces
% (here first order conforming P1 Lagrange functions).
sfun  = {'sflag1';'sflag1'};

% Coefficients to use for each term in the bilinear form.
coef  = [1 1];
i_cub = 3;   % Numerical quadrature rule to use.

[vRowInds,vColInds,vAvals,n_rows,n_cols] = ...
     assemblea( form, sfun, coef, i_cub,  ...
       fea.grid.p, fea.grid.c, fea.grid.a );

% Construct sparse matrix.
A = sparse( vRowInds, vColInds, vAvals, n_rows, n_cols );
```

5. Assemble right hand side load vector by calling *assemblef*

```
% Linear form to assemble (1 = evaluate function values).
form  = [1];

% Finite element shape function.
sfun  = {'sflag1'};
```

```
% Coefficient to use in the linear form.
coef  = [1];

% Numerical quadrature rule to use.
i_cub = 3;

f = assemblef( form, sfun, coef, i_cub, ...
      fea.grid.p, fea.grid.c, fea.grid.a );
```

6. Find indexes to boundary nodes

```
bind = [];
for i_b=1:size(fea.grid.b,2)    % Loop over boundary edges.
  i_c    = fea.grid.b(1,i_b);   % Cell number.
  i_edge = fea.grid.b(2,i_b);
  j_edge = mod( i_edge, size(fea.grid.c,1) ) + 1;
  bind   = [bind fea.grid.c([i_edge j_edge],i_c)'];
end
bind = unique(bind);
```

7. Set homogeneous Dirichlet boundary conditions

```
A = A';   % Transpose matrix for faster row elimination.
A(:,bind) = 0;          % Zero/remove entire rows.
for i=1:length(bind)    % Set diagonal entries of Dirichlet
                        % boundary condition rows to 1.
  i_a        = bind(i);
  A(i_a,i_a) = 1;
end
A = A';   %'

f(bind) = 0;            % Set corresponding RHS entries
                        % to the prescribed boundary values.
```

1. Solve (invert) problem by calling the built-in linear solver

```
u = A\f;
```

2. Create mock fea fields for postprocessing

```
fea.sdim  = { 'x' 'y' };
fea.dvar  = { 'u' };
fea.sfun  = { 'sflag1' };
fea       = parseprob( fea );
```

3. Use postplot to plot the solution

```
fea.sol.u = u;
postplot( fea, 'surfexpr', 'u' )
```

4. Plot the difference between the computed and exact solution

```
u_diff = u - ( 1 - fea.grid.p(1,:)'.^2 - ...
                   fea.grid.p(2,:)'.^2 )/4;
clf
fea.sol.u = u_diff;
postplot( fea, 'surfexpr', 'u' )
```

7.15 Classic Equation - Poisson Equation with a Point Source

The Classic Equation Example - Poisson Equation with a Point Source example in the quickstart guide shows how to model a circle with a point source at its center.

7.16 Classic Equation - Poisson Equation for a Sphere

The following example illustrates how to set up and solve the Poisson equation on a unit sphere $-\Delta u = 1$ ($r = 0..1$) with homogeneous boundary conditions, that is $u = 0|_{r=1}$ with the FEATool FEM assembly primitives.

To derive the finite element discretization of the general Poisson equation $-\Delta u = f$ first multiply the equation with an arbitrary function v (called test function), and integrate over the whole domain Ω, that is $\int_\Omega -\Delta u v \, d\Omega = \int_\Omega f v \, d\Omega$. By applying the Gauss theorem or partial integration to the left side we get

$$\int_\Omega \nabla u \cdot \nabla v \, d\Omega + \int_{\partial\Omega} -\hat{\mathbf{n}} \cdot \nabla u \, dS = \int_\Omega f v \, d\Omega$$

The boundary term can be neglected assuming that we only have prescribed or fixed value (Dirichlet) boundary conditions. In 3D the equation will look like

$$\int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial u}{\partial z}\frac{\partial v}{\partial z} \, dxdydz = \int_\Omega f v \, dxdydz$$

By discretizing the solution (trial function space) variable as $u = \sum_{i=1}^{N} \phi_i u_i$ and similarly test function $v = \sum_{j=1}^{N} \phi_j$, where $\phi_{i/j}$ are the finite element shape or basis functions (usually taken the same for i and j in the Galerkin approximation), and u_i is the value of the solution at node i. Inserting this we get

$$\int \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{\partial \phi_i}{\partial x}\frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_i}{\partial y}\frac{\partial \phi_j}{\partial y} + \frac{\partial \phi_i}{\partial z}\frac{\partial \phi_j}{\partial z} \, u_i = \int \sum_{j=1}^{N} f \phi_j$$

which after discretization and integration with numerical quadrature gives us a matrix system to solve

$$Au = b$$

To start implementing the model pragmatically we first have to create a grid. Instead of creating a grid by using a geometry object we use a geometric sphere grid primitive directly

```
grid = spheregrid();
p = grid.p;   % Extract grid points.
c = grid.c;   % Extract grid cells.
a = grid.a;   % Extract adjacency information.
```

To assemble the matrix **A** we can identify three bi-linear terms (with a coefficient of one) $1 \cdot \phi_{i,x}\phi_{i,y} + 1 \cdot \phi_{i,y}\phi_{j,y} + 1 \cdot \phi_{i,z}\phi_{j,z}$. To specify this in FEATool first create a cell array with coefficient values (or expressions) for each term, in this case `coef = {1 1 1};`. Function values and/or derivatives for the trial and test functions are specified in a 2 by n-terms vector, here `form = [2 3 4;2 3 4];` where the first row specifies trial functions to use for the three terms (1=function value, 2=x-derivative, 3=y-derivative, 4=z-derivative), and the bottom row the test functions. Lastly, the actual shape or basis functions are specified in a cell array as `sfun = {'sflag1';'sflag1'};`, which prescribes first order linear Lagrange shape functions for both the trial and test function spaces. Given a grid with grid points **p** and cell connectivity **c** the call to the fem assembly routine looks like

```
coef   = { 1 1 1 };
form   = [ 2 3 4 ; ...
           2 3 4 ];
sfun   = { 'sflag1' ; ...
           'sflag1' };
i_cub  = 3;              % Numerical quadrature rule.
cind   = 1:size(c,2);    % Index to cells to assemble.
[i,j,s,m,n] = assemblea( form, sfun, coef, i_cub, p, c, a );
A      = sparse( i, j, s, m, n );
```

Similarly, the right hand side linear form or load vector **b** can be assembled as

```
coef   = { 1 };
form   = [ 1 ];
sfun   = { 'sflag1' };
i_cub  = 3;              % Numerical quadrature rule.
cind   = 1:size(c,2);    % Index to cells to assemble.
b      = assemblef( form, sfun, coef, i_cub, p, c, a );
```

Prescribing homogeneous (zero value) Dirichlet boundary conditions on the boundary of the domain can be done by first finding the indexes to the nodes on the boundary (in this case the unit sphere), and then setting the corresponding values in the load vector to zero. Moreover, the corresponding rows in the matrix **A** must also be modified so that they are all zero except for the diagonal entries which are set to one (this preserves the boundary values specified in the **b** vector).

```
radi = sqrt(sum(p.^2,1));        % Radius of grid points.
bind = find(radi>1-sqrt(eps));   % Find grid points
                                 % on the boundary.

b(bind)   = 0;            % Set 'bind' rhs entries to zero.

A(bind,:) = 0;            % Set 'bind' rows to zero.
for i=1:length(bind)      % Set 'bind' diagonal entries to 1
  i_a        = bind(i);
  A(i_a,i_a) = 1;
end
```

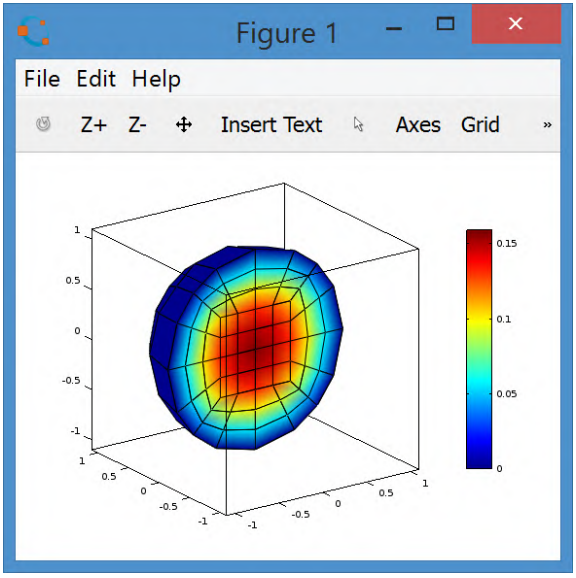
The problem is now fully specified and can be solved as

```
u = A\b;
```

where **u** now contains the values of the solution in the grid points/nodes. To visualize the solution we put everything in a FEATool *fea* struct

```
fea.grid  = grid;
fea.dvar  = { 'u' };
fea.sfun  = sfun;
```

```
fea.sol.u = u;
fea = parseprob( fea );

postplot( fea, 'surfexpr', 'u', 'selcells', 'y>-0.1' )
```



7.17 Other Examples

The following selection of command line examples and test cases can be found in the examples directory.

Field	Description
ex_brinkman1	Coupled porous and incompressible laminar flow.
ex_convdiff1	Convection and diffusion on a unit square.
ex_convdiff2	Periodic convection and diffusion on a line with exact solution.
ex_convdiff3	Infinite time dependent convection and diffusion on a line.
ex_convdiff4	One dimensional Burgers equation with steady solution.
ex_convdiff5	Dominating convection example requiring artificial stabilization.
ex_convdiff6	1D stationary convection-diffusion-reaction with exact solution.
ex_convreact1	Time dependent one dimensional convection and reaction model.
ex_custom_equation1	Black-Scholes model equation implemented as a custom equation.
ex_euler_beam1	1D Euler-Bernoulli beam example.
ex_euler_beam2	1D Euler-Bernoulli beam vibration mode example.
ex_nonlinear_pde1	1D nonlinear PDE with analytical solution.
ex_nonlinear_pde2	1D nonlinear PDE with analytical solution extended to 2D.
ex_diffusion1	Diffusion equation on a unit square with different solutions.

ex_diffusion2	Diffusion equation on a line with exact solutions.
ex_heattransfer1	2D heat conduction with natural convection and radiation.
ex_heattransfer2	One dimensional stationary heat transfer with radiation.
ex_heattransfer3	One dimensional transient heat conduction.
ex_heattransfer4	Two dimensional heat transfer with convective cooling.
ex_heattransfer5	Two dimensional transient cooling shrink fitting example.
ex_heattransfer6	Axisymmetric steady state heat conduction of a cylinder.
ex_heattransfer7	One dimensional transient heat conduction with analytic solution.
ex_laplace1	Laplace equation on a unit square.
ex_laplace2	Laplace equation on a unit circle.
ex_linearelasticity1	Linear elasticity test case, solid stress-strain for a cube.
ex_linearelasticity2	Three dimensional example of stress on a bracket.
ex_linearelasticity3	Parametric study of the bracket deformation model.
ex_linearelasticity4	Stress of loaded I-beam supported by two brackets.
ex_spanner	Gmsh import of a mesh and stress calculation in a spanner.
ex_multiphase1	Multiphase flow example of a static bubble.
ex_multiphase2	Multiphase flow example of a rising bubble.
ex_multiphase3	Multiphase flow example of a breaking dam.
ex_heat_exchanger1	Free and forced convection in a heat exchanger.

ex_natural_convection	Natural convection in a square cavity.
ex_darcy1	Porous media flow in a packed bed reactor using Darcy's law.
ex_navierstokes1	Incompressible fluid flow in a channel.
ex_navierstokes2	Incompressible driven cavity flow.
ex_navierstokes3	Incompressible fluid flow around a cylinder in a channel (Re=20).
ex_navierstokes4	Incompressible fluid flow over a backwards facing step.
ex_navierstokes5	Two dimensional decay of a standing vortex.
ex_navierstokes6	Time dependent flow around a cylinder in a channel (0<=Re<=100).
ex_navierstokes7	Laminar flow in a curved three dimensional pipe.
ex_navierstokes8	Axisymmetric flow in a circular pipe.
ex_navierstokes9	Axisymmetric impinging jet.
ex_navierstokes10	3D flow in a pipe using the OpenFOAM solver.
ex_navierstokes11	3D cavity flow in a cube using the OpenFOAM solver.
ex_navierstokes12	3D flow over a backwards facing step using the OpenFOAM solver.
ex_navierstokes13	3D flow past a cylinder using the OpenFOAM solver.
ex_navierstokes14	2D Pousille flow due to pressure gradient.
ex_swirl_flow1	2D axisymmetric swirl flow in a tube with analytic solution.
ex_swirl_flow2	2D axisymmetric swirl flow in a stepped narrowing tubular region.
ex_swirl_flow3	2D axisymmetric time dependent Taylor-Couette swirl flow.

ex_swirl_flow4	2D axisymmetric swirl flow around a rotating disk.
ex_piezoelectric1	Bending of a beam due to piezoelectric effects.
ex_electrostatics1	Electrostatics test problem.
ex_electrostatics2	Axisymmetric model of a spherical capacitor.
ex_resistive_heating1	Resistive heating of a spiral tungsten filament.
ex_magnetostatics1	Magnetostatics test problem.
ex_magnetostatics2	Magnet field around a horseshoe magnet.
ex_magnetostatics3	Cylindrical magnet without electrical currents.
ex_magnetostatics4	Axisymmetric model of cylindrical magnet.
ex_magnetostatics5	2D Magnetostatics test model.
ex_planestress1	Stress computation for a hole in a thin plate.
ex_planestress2	NAFEMS plane stress benchmark example.
ex_planestress3	NAFEMS plane stress benchmark of a tapered plate.
ex_planestress4	NAFEMS benchmark of plate deformation due to thermal stress.
ex_planestress5	NAFEMS benchmark for plane stress in an elliptic membrane.
ex_planestress6	2D Plane stress analysis of a pressure vessel.
ex_planestrain1	2D Plane strain analysis of a pressure vessel.
ex_axistressstrain1	Axisymmetric stress-strain of a hollow cylinder.
ex_axistressstrain2	Axisymmetric stress-strain of a hollow sphere.
ex_axistressstrain3	Axisymmetric disc with fixed edge and point load.

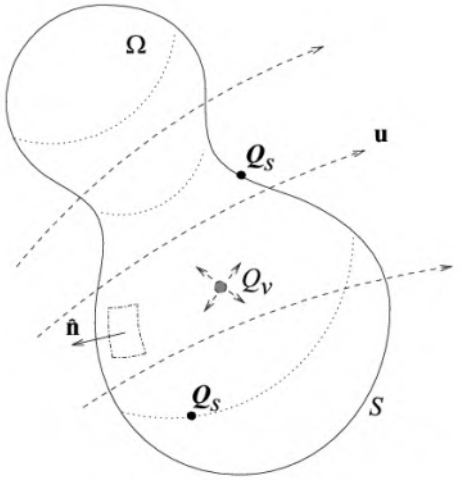
ex_axistressstrain4	Stress formed in brake disk due to heat build up.
ex_poisson1	Poisson equation on a line.
ex_poisson2	Poisson equation on a circle.
ex_poisson3	Poisson equation on a unit square.
ex_poisson4	Poisson equation on a rectangle with complex solution.
ex_poisson5	Poisson equation on a sphere.
ex_poisson6	Poisson equation on a unit cube.
ex_poisson7	Poisson equation on a unit circle with a point constraint.
ex_periodic1	Moving 1D pulse in a periodic domain.
ex_periodic2	2D Periodic Poisson equation example.
ex_robinbc1	1D example with a Robin boundary condition.
ex_waveequation1	Wave equation on a unit circle.

8 Mathematical Modeling

To model physical phenomena such as heat transfer, structural stresses, fluid dynamics, chemical reactions, wave propagation and more, a common approach is the continuum model which formulates the problems as partial differential equations. A typical model PDE can be derived by examining the rate of change of the involved quantities in an infinitesimal volume. This chapter introduces some mathematical theory for to derive FEM approach used by FEATool.

8.1 Conservation Laws

Consider a prototypical property or quantity q, existing in an arbitrary domain Ω. For this quantity to be conserved within Ω, the rate of change of q must be equal to the normal flux crossing the total surface S (let $\hat{\mathbf{n}}$ denote the unit normal to S). With flux meaning the transport of q, in and out of Ω, with a vector field $\mathbf{u}$. Any generation from directional surface sources $\mathbf{Q}_S$ and scalar internal sources Q_V also contributes to an increased (or decreased) amount of q. The following figure illustrates these phenomena for the region Ω.



A balance equation explicitly stating the conservation of q will have the form

$$\frac{\text{net increase of } q}{\text{time}} = \frac{\text{influx}}{\text{time}} + \frac{\text{internal generation}}{\text{time}} + \frac{\text{surface generation}}{\text{time}}$$

which is easy to translate into mathematical vector notation

$$\frac{\partial}{\partial t}\int_\Omega q\,d\Omega = -\int_S q\mathbf{u}\cdot\hat{\mathbf{n}}\,dS + \int_\Omega Q_V\,d\Omega - \int_S \mathbf{Q}_S\cdot\hat{\mathbf{n}}\,dS.$$

Application of Gauss' divergence theorem

$$\int_S \mathbf{F}\cdot\hat{\mathbf{n}}\,dS = \int_\Omega \nabla\cdot\mathbf{F}\,d\Omega$$

to the surface terms results in the following reformulation

$$\frac{\partial}{\partial t}\int_\Omega q\,d\Omega = -\int_\Omega \nabla\cdot q\mathbf{u}\,d\Omega + \int_\Omega Q_V\,d\Omega - \int_\Omega \nabla\cdot\mathbf{Q}_S\,d\Omega.$$

Let us consider the conservation of mass by substituting q with the density ρ. Under the assumption that the system is isolated it is possible to disregard any generation from internal and surface sources. What is left is the following balance equation

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\, d\Omega = -\int_{\Omega}\nabla\cdot(\rho\mathbf{u})\, d\Omega$$

which must hold for a volume of any size. Thus, in the limit $\Omega \to 0$ the well known mass conservation or continuity equation will appear, which reads

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0.$$

Moving on to conservation of linear momentum which, from Newton's second law, is the amount of mass transported with a certain velocity. Specifically, momentum transport per unit volume is therefore expressed as the density ρ transported with the velocity field $\mathbf{u}$. If now the product of these quantities is substituted for q in the general conservation equation then the flux of momentum over the boundaries will accordingly be represented by the quantity $\rho\mathbf{uu}$. Moreover, momentum is created and destroyed by forces acting internally and on the surface of Ω. Inside of Ω momentum is generated by the presence of forces, here labeled $\mathbf{f}$, while on the surface momentum is created by both pressure p and shear stresses, which combined are represented by the stress tensor τ. Having made these observations enables the following formulation for the conservation of momentum

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\mathbf{u}\, d\Omega = -\int_{\Omega}\nabla\cdot(\rho\mathbf{uu})\, d\Omega + \int_{\Omega}\mathbf{f}\, d\Omega + \int_{\Omega}\nabla\cdot\tau\, d\Omega.$$

This should again be valid for a volume of any size or shape with the consequential removal of the integral operations, thus yielding the partial differential equation

$$\frac{\partial}{\partial t}(\rho\mathbf{u}) = -\nabla\cdot(\rho\mathbf{uu}) + \mathbf{f} + \nabla\cdot\tau.$$

The stress tensor τ can, as stated above, be subdivided into two parts, the static pressure p and the shear stresses σ. Consider for the moment Ω in the form of a unit cube where the coordinate axes are aligned with the surface normals. For a given coordinate axis, the pressure term will now only have one component which is acting directly against the surface normal, while the shear stress terms will have three different components, one in each coordinate direction. This enables a splitting of τ into the two terms $p\mathbf{I}$ and σ, that is

$$\frac{\partial}{\partial t}(\rho\mathbf{u}) = -\nabla\cdot(\rho\mathbf{uu}) + \mathbf{f} + \nabla\cdot(-p\mathbf{I} + \sigma)$$

where $\mathbf{I}$ denotes the diagonal unit matrix.

8.2 Poisson Equation

The Poisson equation is a very good model equation to study since it can represent many different physical processes such as diffusion, thermal conduction, and electric potential. The following example illustrates how to set up and solve the Poisson equation on a unit sphere $-D\Delta u = 1$ ($r = 0..1$) with homogeneous boundary conditions, that is $u = 0|_{r=1}$.

To derive the finite element discretization of the general Poisson equation $-\Delta u = f$ first multiply the equation with an arbitrary function v (called test function), and integrate over the whole domain Ω, that is $\int_\Omega -\Delta u v \, d\Omega = \int_\Omega f v \, d\Omega$. By applying the Gauss theorem or partial integration to the left side we get

$$\int_\Omega \nabla u \cdot \nabla v \, d\Omega + \int_{\partial\Omega} -\hat{\mathbf{n}} \cdot \nabla u \, dS = \int_\Omega f v \, d\Omega$$

The boundary term can be neglected assuming that we only have prescribed or fixed value (Dirichlet) boundary conditions. In 2D the equation will look like

$$\int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} \, dxdy = \int_\Omega f v \, dxdy$$

By discretizing the solution (trial function space) variable as $u = \sum_{i=1}^{N} \phi_i u_i$ and similarly test function $v = \sum_{j=1}^{N} \phi_j$, where $\phi_{i/j}$ are the finite element shape or basis functions (usually taken the same for i and j in the Galerkin approximation), and u_i is the value of the solution at node i. Inserting this we get

$$\int \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{\partial \phi_i}{\partial x}\frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_i}{\partial y}\frac{\partial \phi_j}{\partial y} \, u_i = \int \sum_{j=1}^{N} f \phi_j$$

which after discretization and integration with numerical quadrature gives us a matrix system to solve

$$Au = b$$

8.3 Boundary and Initial Conditions

The modeled applications and processes are usually very specific in nature and the studies can thus be confined to a smaller spatial sub region or domain $\Omega \subset \mathbf{R}^d$ and a specific temporal extent $[0, T]$. Both these restrictions will usually greatly simplify the modeling and also reduce the effort required to obtain a solution. It is particularly advantageous to utilize all existing symmetry axes to further shrink the computational domain. Sometimes it is also possible to transform a three dimensional model to two dimensions if the problem can be considered axisymmetric. The drawback of restricting the models spatially is that the equations now have to be supplied with suitable boundary conditions, which goal is to describe all interactions between Ω and the rest of the non-modeled environment.

Boundary conditions usually come in two distinct variants, Dirichlet conditions which fixes the value of a quantity

$$u = r \quad on \quad \partial\Omega_D.$$

and Neumann, also called natural, boundary conditions which specify the in- or outward-flux

$$-\hat{\mathbf{n}} \cdot D\nabla u = g \quad on \quad \partial\Omega_N.$$

Initial conditions, in the form of specified $u(\mathbf{x}, 0) = u_0(\mathbf{x})$, must be prescribed in addition to the boundary conditions for applications where the temporal evolution is of interest. For nonlinear stationary calculations it is also advantageous to specify a good initial guess so that the nonlinear solver will converge faster.

8.4 Poisson Example

A tutorial example for the Poisson equation on a circular domain can be found in the Classic Equation - Poisson Equation on a Circle tutorial section. This example is split up into four parts showing how set up and solve the model in the GUI, as well as on the command line using physics modes, and the Poisson Equation using the CLI and core FEM library functions FEM core library functions such as matrix assembly.

9 Problem Definition

FEATool uses a struct format with several fields to specify and set up problem definitions. The FEATool GUI will automatically manage these fields, however if the model is exported, modified on the command line, or built from scratch it is important to know how these fields are used, which is explained below.

9.1 Problem Definition Struct

The FEATool problem definition struct consists of the following main fields

Field	Description
geom	Geometry objects
grid	Grid structure
sdim	Cell array of space dimension names
phys	Physics mode structs
dvar	Cell array of dependent variable names
sfun	Cell array of shape functions
coef	Cell array of coefficient and expression definitions

eqn	Equation specification struct
bdr	Boundary specification struct
pnt	Point source and constraint specification struct
sol	Solution vector and list of solution times

A cell array of geometry objects used for grid generation is contained in the **geom.objects** field. The geometry is only used by the automatic grid generation and can be omitted if a grid is constructed by other means (for example with the grid function primitives). For example, the following code defines a geometry with a unit circle

```
fea.geom.objects = { gobj_circle() };
```

The **grid** struct field, which defines the computational grid, is explained in the Grid section.

The **sdim** field contains a cell array of strings with names for the space dimensions. These can then be used in FEATool functions, and equation/boundary specification and postprocessing. Typically the field will take the following forms

```
fea.sdim = { 'x' };           % 1D.
fea.sdim = { 'x' 'y' };       % 2D.
fea.sdim = { 'r' 'z' };       % 2D - Axisymmetry.
fea.sdim = { 'x' 'y' 'z' };   % 3D.
```

but the name strings can be substituted arbitrarily.

The optional **phys** struct contains predefined physics modes that have been added with the addphys function. After editing and setting the appropriate physics mode coefficients the parsephys command will expand the physics to the fea parent **dvar**, **sfun**, **coef**, **eqn**, and **bdr** fields.

The **dvar** field contains a cell array of strings with names of the dependent (equation) variables which should be solved for. For example

```
fea.dvar = { 'u' 'v' };
```

defines two dependent variables, labeled *u* and *v*.

Similarly, the **sfun** field contains a cell array of strings with function names of the finite element shape functions to use for each dependent variable. Shape functions are defined in the *ellib* directory. For example

```
fea.dvar = { 'sflag1' 'sflag2' };
```

defines 1st order conforming basis functions for the first dependent variable and 2nd order for the second one.

The **coef** field is a (n_coef, 4) sized cell array of coefficient expressions that can be used in equation definitions and postprocessing. The first column gives the coefficient names, second a short description, third a long description, and the fourth column is a sub cell array for the coefficient expression definitions in each subdomain. (If any of the fields *const*, *expr*, and *vars* exist they will be merged internally with the *coef* field.) The assignment

```
fea.coef = { 'rho' '' '' { 1  2           } ;
             'miu' '' '' { 3 '4*pi*sin(x)' } };
```

here defines two coefficients *rho* with values *1* and *2* in two respective subdomains, and *miu* with values *3* and the expression *4∗pi∗sin(x)*.

The **eqn**, **bdr**, and **pnt** structs are used to specify equations, boundary conditions, and point constraints. The composition of these fields is explained below in the corresponding subsections.

After solving a problem the **sol** field will contain the solution column vector **u** with rows corresponding to the degrees of freedom (ordered according to the order of the dependent variables in *fea.dvar*). For time dependent problems the columns in **u** correspond to solutions at different times, and additionally the output times will be stored in the **tlist** vector.

9.1.1 Equation Struct Specification

The equation struct contains the following fields

Field	Description
eqn.m.form	Form specifications for temporal terms
eqn.m.coef	Coefficient specifications for temporal terms
	Shape functions are inherited from eqn.a.sfun

Field	Description
eqn.a.form	Bilinear form specifications
eqn.a.coef	Coefficient specifications for bilinear forms
eqn.a.sfun	Shape function specification for bilinear forms

Field	Description
eqn.f.form	Linear form specifications for right hand side/load vector
eqn.f.coef	Coefficient specifications for right hand side/load vector
eqn.f.sfun	Shape function specifications for right hand side/load vector

Field	Description
eqn.dofm	Degree of freedom n numbering map for each cell
eqn.ndof	Numbers of degrees of freedom for each dependent variable

The **eqn.m** field contains specifications for the time dependent term (with time derivative). Similarly the **eqn.a** field contains specifications for the bilinear forms used in the iteration (stiffness) matrix, and the **eqn.f** field specifies the linear forms in the right hand side/load vector.

The above struct fields contain the **form** field which specifies the (bi-)linear forms to build and assemble. For bilinear forms the first row corresponds to the trial function space, and the second row the test function space. Linear forms only need to contain one row. In the form specification a 1 indicates a function value, 2 x-derivative, 3 y-derivative, and 4 z-derivative. For example a form specification [2 3;2 3] indicates a bilinear form with two terms, one term with both x-derivatives for the test and trial function spaces, and one with y-derivatives for both spaces (which in this case is a typical two-dimensional diffusion operator). Second order derivatives can also be specified as 22 for the xx-derivative, 23 xy-derivative and so on.

The **coef** field, is a cell array with coefficient values or expressions used for each term in the **form** field.

The **sfun** field is a cell array of shape function names used in the form assembly. This field is usually automatically constructed when calling the parseprob function.

dofm is an array specifying the local to global degree of freedom numbering for each dependent variable. The rows correspond to local degrees of freedoms on each cell and the columns give the cell numbers. (For linear conforming shape functions the dof mapping will be identical to the grid.c field.) This field is created when parseprob calls the mapdofbdr function.

ndof is simply an help array for the numbers of degrees of freedom for each dependent variable (equals to max(dofm(:)) and also automatically generated by mapdofbdr).

9.1.2 Boundary Struct Specification

The boundary struct contains the following fields

Field	Description
bdr.d	Dirichlet boundary coefficients
bdr.n	Neumann (flux) boundary coefficients
bdr.bdrm	Boundary degree of freedom numbering maps

Dirichlet boundary conditions are used to prescribe and fix a specific value for the dependent variables, while in contrast Neumann conditions represent a inward or outward directed flux (which are functions of the gradients of the dependent variables). Mixed Robin conditions are currently not available in FEATool. If a Dirichlet condition is prescribed on a boundary, the corresponding Neumann flux entry will be ignored. Alternatively, if a Dirichlet boundary coefficient entry is empty the Neumann flux contribution will be computed and added (however, all default Neumann contributions are zero corresponding to homogeneous do-nothing Neumann boundary conditions).

Dirichlet boundary coefficient values or string expressions for each boundary segment are specified in a cell array in the **bdr.d** field. Two syntaxes are supported, standard syntax where **d** is a cell array of size *n_dvar x n_bdr*, or extended syntax (supported in versions 1.5 or later) with **d** a *1 x n_dvar* cell array where each entry itself is a *n_bc_groups x n_bdr* cell array containing the coefficients. In the extended syntax the number of boundary condition groups *n_bc_groups* is due to the chosen finite element shape function (specifically the number of *nLDof* rows which is *1+n_sdim* for Hermite elements and *1* for all others). The extended syntax is used to prescribe separate conditions on the different local degrees of freedom (that is the basis function values and derivatives for Hermite elements). The Dirichlet boundary conditions are prescribed by the function bdrsetd.

If a Dirichlet boundary coefficient entry is empty, then Neumann (flux) conditions will be prescribed instead for the corresponding boundary segment. Similar to Dirichlet conditions, Neumann conditions are specified in the **n** field in either standard or extended syntax (bdrneu). An empty cell indicates homogeneous Neumann condition to be used (zero flux) for that boundary segment.

The **bdrm** arrays specifies the boundary condition mapping for each dependent variable. In an **bdrm** array the first row gives the global degree of freedom number, second the cell number, third cell edge number, fourth boundary section number, and fifth local degree of freedom number. While the **d** and **n** fields must be prescribed by the user or derived from the physics

modes with a call to parsephys, the **bdrm** field is automatically created when parseprob calls the mapdofbdr function.

For example, assume a two dimensional time dependent problem with one dependent variable or unknown *u*, then the following standard boundary syntax

```
fea.bdr.d = {  1  'sin(pi*t*x)+u'  []       []       };
fea.bdr.n = { []         0          2  'nx*ux+ny*uy' };
```

will prescribe *u* constant *u = 1* on the first boundary, a string expression *u(x,t) = sin(pi∗t∗x)+u* on the second boundary, a constant flux Neumann condition *2* on the third boundary, and finally the flux expression *nx∗ux+ny∗uy* on the fourth and last boundary. Note that here in the string expressions it is perfectly valid to use constants, Matlab and Octave functions, time *t*, function values and derivatives of the dependent variables, and also the normals, here *nx* and *ny*.

Assuming using a finite element basis functions with only one boundary condition group the equivalent extended syntax looks like the following

```
fea.bdr.d = { {  1  'sin(pi*t*x)+u'  []       []       } };
fea.bdr.n = { { []         0          2   'nx*ux+ny*uy' } };
```

If the default solvers *solvestat* and *solvetime* detects nonlinear Neumann boundary conditions they will attempt to linearize them by moving terms involving linear forms from the explicit right hand side to the implicit matrix (a boundary contribution to *eqn.a* instead of *eqn.f*). This is done by the bdrneu_lin function.

9.1.3 Point Struct Specification

The point struct contains the following fields

Field	Description
pnt.index	Index to grid point (in grid.p)
pnt.type	Specifies either sources or constraint
pnt.dvar	Integer or string corresponding a dependent variable
pnt.expr	scalar or string expression for constraint

Point sources and Dirichlet point constraints can be specified in a **pnt** struct. The **type** field specifies either a point 'source' or 'constraint' for each point with a corresponding string value. Point sources are applied to the right hand side load vector *f* while constraints are applied to both *f* and the global matrix *A* with the information in the finite element problem struct (pntsetf and pntset). The sources or constraints will be applied to the degree of freedom closest to the grid point specified in the **index** field. The **dvar** field contains an integer or string expression pointing to a dependent variable in **dvar**. The sources or constraints are specified in the **expr** field either as a string expression or scalar so that *pnt.dvar(grid.p(:,pnt.index)) = pnt.expr*. (Note that a point constraint will override the usual Dirichlet boundary conditions at the given point).

10 Function Index

10.1 File List

Here is a list of all files with brief descriptions:

featool/addpaths.m
ADDPATHS Add/remove FEATool directories to/from working paths 275
featool/evalexpr.m
EVALEXPR Evaluates an expression in points 520
featool/featool.m
FEATOOL Start the FEATool Multiphysics GUI 749
featool/gridgen.m
GRIDGEN Grid generation for geometry objects 802
featool/install.m
INSTALL Add the FEATool toolbox to the Octave/Matlab path 813
featool/intbdr.m
INTBDR Integation of expression over boundaries 813
featool/intsubd.m
INTSUBD Integate expression over subdomains 813
featool/minmaxbdr.m
MINMAXBDR Calculate boundary minima and maxima 815
featool/minmaxsubd.m
MINMAXSUBD Calculate subdomain minima and maxima 814
featool/plotbdr.m
PLOTBDR Boundary plot function 818

featool/plotgeom.m
PLOTGEOM Visualization of geometry objects . . . 819
featool/plotgrid.m
PLOTGRID Grid plot function . . . 819
featool/plotsubd.m
PLOTSUBD Subdomain plotting function . . . 820
featool/postplot.m
POSTPLOT Postprocessing and visualization function . . . 824
featool/solvestat.m
SOLVESTAT Solve stationary problem . . . 825
featool/solvetime.m
SOLVETIME Solve time-dependent problem . . . 826
featool/uninstall.m
UNINSTALL Remove the FEATool toolbox from the Octave/Matlab path . . . 827
featool/core/asmgroups.m
ASMGROUPS Determine cell indices for assembly subdomian/cell groups 275
featool/core/asmrowcolptr.m
ASMROWCOLPTR Compute row and column pointers for sparse matrix assembly . . . 276
featool/core/assemblea.m
ASSEMBLEA Finite element assembly of bilinear forms . . . 276
featool/core/assemblef.m
ASSEMBLEF Finite element assembly of linear form (rhs/load vector) . . . 278
featool/core/assemblej.m
ASSEMBLEJ Finite element assembly of Jacobian forms . . . 279
featool/core/assemblenjac.m
ASSEMBLENJAC Assembly of defect and Newton Jacobian . . . 281
featool/core/assembleprob.m
ASSEMBLEPROB Assembly of system matrix and right hand side/load vector . . . 281
featool/core/assemjac.m
ASSEMJAC Assemble monolithic Jacobian matrix . . . 282
featool/core/assemmat.m
ASSEMMAT Assemble monolithic matrix . . . 283
featool/core/bdrcpl.m
BDRCPL Set coupled Dirichlet boundary conditions . . . 285

featool/core/bdrneu.m
BDRNEU Compute Neumann (flux) boundary condition vector 284
featool/core/bdrneu_lin.m
BDRNEU_LIN Linearize Neumann boundary contributions 284
featool/core/bdrsetd.m
BDRSETD Set Dirichlet boundary conditions 285
featool/core/bdrslip.m
BDRDLIP Set slip boundary conditions 286
featool/core/cellvol.m
CELLVOL Computes cell volume for cells 286
featool/core/coordr2l.m
COORDR2L Convert real to local coordinates 286
featool/core/coordtf.m
COORDTF Transforms (interpolates) coordinates from local to real coordinates 287
featool/core/evaldvar.m
EVALDVAR Evaluates a dependent variable in a point on a group of cells . 289
featool/core/evalexpr0.m
EVALEXPR0 Evaluates an expression in a point on a group of cells 289
featool/core/evalexprc.m
EVALEXPRC Evaluates an expression in cells 290
featool/core/evalexprp.m
EVALEXPRP Evaluates an expression in the grid points 290
featool/core/evalinit.m
EVALINIT Calculates initial solution vector from given expressions 291
featool/core/evalsfun.m
EVALSFUN Shape function evaluation driver subroutine 291
featool/core/lcoorde.m
LCOORDE Get local coordinates on a cell edge/face 292
featool/core/mapdofbdr.m
MAPDOFBDR Create degree of freedom and boundary maps 292
featool/core/pntset.m
PNTSET Set Dirichlet point constraints 294
featool/core/pntsetf.m
PNTSETF Assemble point source terms 294

featool/core/quadefc.m
QUADEFC Get local quadrature coordinates on edges/faces 288
featool/core/quadrule.m
QUADRULE Calculate numerical quadrature/integration rules 287
featool/core/quadruleb.m
QUADRULEB Calculates numerical integration rules for cell boundaries . 288
featool/core/re2locquad.m
RE2LOCQUAD Convert quadrialteral real to local coordinates 293
featool/core/solve_hook.m
SOLVE_HOOK Custom solver hook calling function 293
featool/core/subcoef.m
SUBCOEF Substitute coefficients from cell array 295
featool/core/tfjac.m
TFJAC Computes transformation Jacobian and determinant 295
featool/core/tfjachex.m
TFJACHEX Computes transformation Jacobian inverse and determinant for hexahedrals 296
featool/core/tfjacline.m
TFJACLINE Computes transformation Jacobian inverse and determinant for lines 297
featool/core/tfjacquad.m
TFJACQUAD Computes transformation Jacobian and determinant for quadrilaterals 297
featool/core/tfjactet.m
TFJACTET Computes transformation Jacobian inverse and determinant for tetraheda 298
featool/core/tfjactri.m
TFJACTRI Computes transformation Jacobian inverse and determinant for triangles 299
featool/ellib/grid_cell_type.m
GRID_CELL_TYPE Determine grid cell type 519
featool/ellib/lcoord_cell.m
LCOORD_CELL Local cell coordinates for the degrees of freedom 515
featool/ellib/sf_disc0.m
SF_DISC0 Constant discontinuous shape function 299

featool/ellib/sf_disc1.m
SF_DISC1 Linear discontinuous shape function (P1) 301
featool/ellib/sf_hex_Q1.m
SF_HEX_Q1 Trilinear conforming shape function for hexahedrons (Q1) . . 302
featool/ellib/sf_hex_Q1nc.m
SF_HEX_Q1NC Trilinear nonconforming shape function for hexahedrons (Q1∼) 307
featool/ellib/sf_hex_Q2.m
SF_HEX_Q2 Triquadratic conforming shape function for hexahedrons (Q2) 312
featool/ellib/sf_hex_Q3.m
SF_HEX_Q3 Tricubic conforming shape function for hexahedrons (Q3) . . 323
featool/ellib/sf_hex_Q4.m
SF_HEX_Q4 Triquartic conforming shape function for hexahedrons (Q4) . 332
featool/ellib/sf_hex_Q5.m
SF_HEX_Q5 Triquintic conforming shape function for hexahedrons (Q5) . 349
featool/ellib/sf_line_H3.m
SF_LINE_H3 Third order 1D C1 Hermite shape functions for lines 376
featool/ellib/sf_line_P3.m
SF_LINE_P3 1D Third order Lagrange shape functions for lines (P3) 378
featool/ellib/sf_line_P4.m
SF_LINE_P4 1D Fourth order Lagrange shape functions for lines (P4) . . . 380
featool/ellib/sf_line_P5.m
SF_LINE_P5 1D Fifth order Lagrange shape functions for lines (P5) 383
featool/ellib/sf_quad_H3.m
SF_QUAD_H3 Third order 2D C1 Hermite shape function for quadrilaterals 385
featool/ellib/sf_quad_Q1.m
SF_QUAD_Q1 Bilinear conforming shape function for quadrilaterals (Q1) . 389
featool/ellib/sf_quad_Q1nc.m
SF_QUAD_Q1NC Bilinear nonconforming shape function for quadrilaterals (Q1∼) 392
featool/ellib/sf_quad_Q2.m
SF_QUAD_Q2 Biquadratic conforming shape function for quadrilaterals (Q2) 395
featool/ellib/sf_quad_Q3.m
SF_QUAD_Q3 Bicubic conforming shape function for quadrilaterals (Q3) . 398

featool/ellib/sf_quad_Q4.m
SF_QUAD_Q4 Biquartic conforming shape function for quadrilaterals (Q4) 402
featool/ellib/sf_quad_Q5.m
SF_QUAD_Q5 Biquintic conforming shape function for quadrilaterals (Q5) 406
featool/ellib/sf_simp_P1.m
SF_SIMP_P1 Linear Lagrange shape function for simplices (P1) 411
featool/ellib/sf_simp_P1nc.m
SF_SIMP_P1NC Linear nonconforming shape function for simplices (P-1) . 412
featool/ellib/sf_simp_P2.m
SF_SIMP_P2 Quadratic Lagrange shape function for simplices (P2) 413
featool/ellib/sf_simp_P2bub.m
SF_SIMP_P2BUB Quadratic Lagrange shape function for simplices with bubble (P2+) 416
featool/ellib/sf_tet_P3.m
SF_TET_P3 Third order Lagrange shape functions for tetrahedra (P3) . . . 420
featool/ellib/sf_tet_P4.m
SF_TET_P4 Fourth order Lagrange shape functions for tetrahedra (P4) . . 432
featool/ellib/sf_tet_P5.m
SF_TET_P5 Fifth order Lagrange shape functions for tetrahedra (P5) . . . 452
featool/ellib/sf_tri_H3.m
SF_TRI_H3 Third order 2D C1 Hermite shape functions for triangles 481
featool/ellib/sf_tri_P3.m
SF_TRI_P3 Third order Lagrange shape functions for triangles (P3) 484
featool/ellib/sf_tri_P4.m
SF_TRI_P4 Fourth order Lagrange shape functions for triangles (P4) . . . 489
featool/ellib/sf_tri_P5.m
SF_TRI_P5 Fifth order Lagrange shape functions for triangles (P5) 496
featool/ellib/sflag1.m
SFLAG1 Shape function driver (first order P1/Q1 Lagrange polynomials) . . 505
featool/ellib/sflag2.m
SFLAG2 Shape function driver (second order P2/Q2 Lagrange polynomials) 506
featool/ellib/sflag3.m
SFLAG3 Shape function driver (third order P3/Q3 Lagrange polynomials) . 507
featool/ellib/sflag4.m
SFLAG4 Shape function driver (fourth order P4/Q4 Lagrange polynomials) 509

featool/ellib/sflag5.m
SFLAG5 Shape function driver (fifth order P5/Q5 Lagrange polynomials) . 511
featool/ellib/sfnoncon1.m
SFNONCON1 Shape function driver (first order P1/Q1 Lagrange polynomials) 512
featool/ellib/sfnonconp1.m
SFNONCONP1 Shape function driver (first order P1 Lagrange polynomials) 513
featool/examples/ex_axistressstrain1.m
EX_AXISTRESSSTRAIN1 Example for hollow cylider axisymmetric stress-strain 706
featool/examples/ex_axistressstrain2.m
EX_AXISTRESSSTRAIN2 Example for a pressurized hollow sphere axisymmetric stress-strain 708
featool/examples/ex_axistressstrain3.m
EX_AXISTRESSSTRAIN3 Disc with fixed edge and central point load axisymmetric stress-strain 710
featool/examples/ex_axistressstrain4.m
EX_AXISTRESSSTRAIN4 Axisymmetric heat induced stress for a brake disk 713
featool/examples/ex_brinkman1.m
EX_BRINKMAN1 Coupled Navier-Stokes and Brinkman equations model . 521
featool/examples/ex_convdiff1.m
EX_CONVDIFF1 2D Convection and diffusion equation example on a rectangle 523
featool/examples/ex_convdiff2.m
EX_CONVDIFF2 1D Time dependent convection and diffusion equation example 526
featool/examples/ex_convdiff3.m
EX_CONVDIFF3 1D Time dependent convection and diffusion equation example 529
featool/examples/ex_convdiff4.m
EX_CONVDIFF4 1D Burgers equation (convection and diffusion) example . 531
featool/examples/ex_convdiff5.m
EX_CONVDIFF5 2D Convection and diffusion equation with high Peclet number 534
featool/examples/ex_convdiff6.m
EX_CONVDIFF6 1D Stationary convection and diffusion equation example 536

featool/examples/ex_convreact1.m
EX_CONVREACT1 1D Time dependent convection-reaction equation example . . . 539
featool/examples/ex_custom_equation1.m
EX_CUSTOM_EQUATION1 1D Black-Scholes custom equation example . . 542
featool/examples/ex_darcy1.m
EX_DARCY1 Darcy's law for a porous packed bed reactor 612
featool/examples/ex_diffusion1.m
EX_DIFFUSION1 2D Diffusion equation example on a unit square 554
featool/examples/ex_diffusion2.m
EX_DIFFUSION2 1D Time dependent diffusion equation examples 557
featool/examples/ex_electrostatics1.m
EX_ELECTROSTATICS1 Electrostatic test example 670
featool/examples/ex_electrostatics2.m
EX_ELECTROSTATICS2 Axisymmetric model of a spherical capacitor . . . 672
featool/examples/ex_euler_beam1.m
EX_EULER_BEAM1 1D Euler-Bernoulli beam model example 544
featool/examples/ex_euler_beam2.m
EX_EULER_BEAM2 1D Euler-Bernoulli beam vibration example 547
featool/examples/ex_heat_exchanger1.m
EX_FREE_CONVECTION Example of free and forced convection around a heated cylinder 605
featool/examples/ex_heattransfer1.m
EX_HEATTRANSFER1 2D ceramic strip with radiation and convection . . . 560
featool/examples/ex_heattransfer2.m
EX_HEATTRANSFER2 1D Stationary heat transfer with radiation 563
featool/examples/ex_heattransfer3.m
EX_HEATTRANSFER3 1D Transient heat conduction 565
featool/examples/ex_heattransfer4.m
EX_HEATTRANSFER4 2D Heat transfer with convective cooling 568
featool/examples/ex_heattransfer5.m
EX_HEATTRANSFER5 2D Transient cooling shrink fitting example 570
featool/examples/ex_heattransfer6.m
EX_HEATTRANSFER6 2D axisymmetric heat conduction 573
featool/examples/ex_heattransfer7.m
EX_HEATTRANSFER7 1D Transient heat diffusion with analytic solution . . 576

featool/examples/ex_laplace1.m
EX_LAPLACE1 2D Laplace equation example on a unit square 578
featool/examples/ex_laplace2.m
EX_LAPLACE2 2D Laplace equation on a circle with nonzero boundary conditions 581
featool/examples/ex_linearelasticity1.m
EX_LINEARELASTICITY1 Example for solid stress-strain on a cube 583
featool/examples/ex_linearelasticity2.m
EX_LINEARELASTICITY2 Example for deflection of a bracket 586
featool/examples/ex_linearelasticity3.m
EX_LINEARELASTICITY3 Parametric study for the bracket deflection model 589
featool/examples/ex_linearelasticity4.m
EX_LINEARELASTICITY4 Stress calculation of an I-beam attached to two brackets 591
featool/examples/ex_magnetostatics1.m
EX_MAGNETOSTATICS1 Magnetostatic test example 677
featool/examples/ex_magnetostatics2.m
EX_MAGNETOSTATICS2 Magnetic field around a horseshoe magnet 679
featool/examples/ex_magnetostatics3.m
EX_MAGNETOSTATICS3 3D Cylindrical magnet example 682
featool/examples/ex_magnetostatics4.m
EX_MAGNETOSTATICS4 2D Axisymmetric cylindrical magnet example . . 683
featool/examples/ex_magnetostatics5.m
EX_MAGNETOSTATICS5 Magnetic field test model 685
featool/examples/ex_multiphase1.m
EX_MULTIPHASE1 Static bubble example 596
featool/examples/ex_multiphase2.m
EX_MULTIPHASE2 Multiphase flow rising bubble example 599
featool/examples/ex_multiphase3.m
EX_MULTIPHASE3 Breaking dam multiphase flow example 602
featool/examples/ex_natural_convection.m
EX_NATURAL_CONVECTION 2D Example for natural convection of air in a square cavity 608
featool/examples/ex_navierstokes1.m
EX_NAVIERSTOKES1 2D Example for incompressible stationary flow in a channel 614

featool/examples/ex_navierstokes10.m
EX_NAVIERSTOKES10 3D Example for stationary flow in a pipe 643
featool/examples/ex_navierstokes11.m
EX_NAVIERSTOKES11 3D Example flow in a cubcic cavity 645
featool/examples/ex_navierstokes12.m
EX_NAVIERSTOKES12 3D Example flow over a backwards facing step . . . 647
featool/examples/ex_navierstokes13.m
EX_NAVIERSTOKES13 3D Example flow past a cylinder 649
featool/examples/ex_navierstokes14.m
EX_NAVIERSTOKES14 2D Example for flow in a channel driven by pressure difference 652
featool/examples/ex_navierstokes2.m
EX_NAVIERSTOKES2 2D Example for incompressible flow in a square cavity 618
featool/examples/ex_navierstokes3.m
EX_NAVIERSTOKES3 2D Example for incompressible stationary flow around a cylinder 622
featool/examples/ex_navierstokes4.m
EX_NAVIERSTOKES4 2D Example for incompressible flow over a backwards facing step 626
featool/examples/ex_navierstokes5.m
EX_NAVIERSTOKES5 2D Vortex flow with analytical solution 630
featool/examples/ex_navierstokes6.m
EX_NAVIERSTOKES6 2D incompressible time-dependent flow around a cylinder 633
featool/examples/ex_navierstokes7.m
EX_NAVIERSTOKES7 3D Example for incompressible stationary flow in a curved pipe 636
featool/examples/ex_navierstokes8.m
EX_NAVIERSTOKES8 2D Example for axisymmetric incompressible stationary flow in a constricted circular pipe 638
featool/examples/ex_navierstokes9.m
EX_NAVIERSTOKES9 2D Axisymmetric jet impingement with heat transfer 640
featool/examples/ex_nonlinear_pde1.m
EX_NONLINEAR_PDE1 1D nonlinear PDE example 549

featool/examples/ex_nonlinear_pde2.m
EX_NONLINEAR_PDE2 2D nonlinear PDE example 551
featool/examples/ex_periodic1.m
EX_PERIODIC1 1D Example of pulse in a periodic line 739
featool/examples/ex_periodic2.m
EX_PERIODIC2 2D Periodic Poisson equation example 741
featool/examples/ex_piezoelectric1.m
EX_PIEZOELECTRIC1 Piezoelectric bending of a beam example 666
featool/examples/ex_planestrain1.m
EX_PLANESTRAIN1 Plane strain analysis of a pressure vessel 703
featool/examples/ex_planestress1.m
EX_PLANESTRESS1 Example for plane stress for hole in plate 687
featool/examples/ex_planestress2.m
EX_PLANESTRESS2 NAFEMS benchmark challenge 1 plane stress example 690
featool/examples/ex_planestress3.m
EX_PLANESTRESS3 NAFEMS benchmark model for stress of a tapered membrane 692
featool/examples/ex_planestress4.m
EX_PLANESTRESS4 Example of thermally induced stress 695
featool/examples/ex_planestress5.m
EX_PLANESTRESS5 Plane stress example for an elliptic membrane 698
featool/examples/ex_planestress6.m
EX_PLANESTRESS6 Plane stress analysis of a pressure vessel 700
featool/examples/ex_poisson1.m
EX_POISSON1 1D Poisson equation example 718
featool/examples/ex_poisson2.m
EX_POISSON2 2D Poisson equation example on a circle 722
featool/examples/ex_poisson3.m
EX_POISSON3 2D Poisson equation example on a unit square 725
featool/examples/ex_poisson3_derivative.m
EX_POISSON3_DERIVATIVE Derivative interpolation function 728
featool/examples/ex_poisson4.m
EX_POISSON4 2D Poisson equation example on a rectangle 728
featool/examples/ex_poisson5.m
EX_POISSON5 3D Poisson equation example on a unit sphere 731

featool/examples/ex_poisson6.m
EX_POISSON6 3D Poisson equation example on a unit cube 734
featool/examples/ex_poisson7.m
EX_POISSON7 Poisson equation on a unit circle with a point source . . . 737
featool/examples/ex_resistive_heating1.m
EX_RESISTIVE_HEATING1 Resistive heating of a tungsten filament 674
featool/examples/ex_robinbc1.m
EX_ROBINBC1 1D Robin boundary condition example 744
featool/examples/ex_spanner.m
EX_SPANNER Gmsh grid import and stress calculation of a spanner 594
featool/examples/ex_swirl_flow1.m
EX_SWIRL_FLOW1 2D Axisymmetric laminar swirl flow 655
featool/examples/ex_swirl_flow2.m
EX_SWIRL_FLOW2 2D Axisymmetric swirl flow in step domain 658
featool/examples/ex_swirl_flow3.m
EX_SWIRL_FLOW3 2D Axisymmetric Taylor-Couette (swirl) flow 660
featool/examples/ex_swirl_flow4.m
EX_SWIRL_FLOW4 2D Rotating swirling flow around a disk 663
featool/examples/ex_waveequation1.m
EX_WAVEEQUATION1 2D Wave equation example on a circle 745
featool/fenics/fenics.m
FENICS Exports, solves, and imports a FEniCS project 749
featool/fenics/fenics_gui.m
FENICS_GUI Fenics project gui control dialog box 750
featool/geom/compute_distance.m
Compute_distance Evaluate distance function 752
featool/geom/extract_entries.m
EXTRACT_ENTRIES Extract array entries and recompute pointers 752
featool/geom/geom_add_gobj.m
GEOM_ADD_GOBJ Add geometry object to geom or fea struct 752
featool/geom/geom_boundary_edges.m
GEOM_BOUNDARY_EDGES Constructs boundary edges from faces 752
featool/geom/geom_combine_bbox.m
GEOM_COMBINE_BBOX Combine bounding boxes 753

featool/geom/geom_decompose_boundaries2.m
GEOM_DECOMPOSE_BOUNDARIES2 Decomposes intersecting boundaries of geometry objects . . . 753
featool/geom/geom_decompose_boundaries3.m
GEOM_DECOMPOSE_BOUNDARIES3 Decomposes intersecting boundaries of geometry objects . . . 753
featool/geom/geom_get_tags.m
GEOM_GET_TAGS Get geometry object tags . . . 753
featool/geom/geom_intersect_lines.m
GEOM_INTERSECT_LINES Compute intersection between two lines . . . 753
featool/geom/geom_parse_formula.m
GEOM_PARSE_FORMULA Parse geom formula . . . 754
featool/geom/geom_restrict_bbox.m
GEOM_RESTRICT_BBOX Restict bounding box . . . 754
featool/geom/geom_split_lines.m
Geom_split_lines Split intersecting lines . . . 754
featool/geom/geom_split_object.m
GEOM_SPLIT_OBJECT Split composite geometry object . . . 755
featool/geom/geom_uniqify_tag.m
GEOM_UNIQIFY_TAG Uniquify geometry object tag . . . 755
featool/geom/geom_update_tags.m
GEOM_UPDATE_TAGS Update the geometry tags field . . . 755
featool/geom/gobj_boundary_vertbbox.m
GOBJ_BOUNDARY_VERTBBOX Compute vertices and bbox from boundaries . . . 755
featool/geom/gobj_reorient_boundaries.m
GOBJ_REORIENT_BOUNDARIES Compute vertices and bbox from boundaries . . . 755
featool/geom/order_boundaries.m
ORDER_BOUNDARIES Determine connected boundaries in order . . . 756
featool/geom/plotgeom1.m
PLOTGEOM1 Visualize 1d geometries . . . 756
featool/geom/plotgeom2.m
PLOTGEOM2 Visualize 2d geometries . . . 756
featool/geom/plotgeom3.m
PLOTGEOM3 Visualize 3d geometries . . . 757

featool/geom/plotgeomb.m
PLOTGEOMB Visualization of geometry boundaries 758
featool/geom/plotpoint.m
PLOTPOINT Visualization of point geometry object 758
featool/geom/polyarea3.m
POLYAREA3 Area of a polygon in 3D space 759
featool/geom/reconstruct_boundaries.m
RECONSTRUCT_BOUNDARIES Reconstruct boundaries 759
featool/geom/reconstruct_triangles.m
RECONSTRUCT_TRIANGLES Reconstruct triangles from point cloud . . . 759
featool/geom2/bbox_intersect.m
BBOX_INTERSECT Compute and test if bounding boxes intersect 759
featool/geom2/boundary_retriangulation.m
BOUNDARY_RETRIANGULATION Retriangulate 3D boundaries 759
featool/geom2/compress_boundary_edges.m
COMPRESS_BOUNDARY_EDGES Compress boundary edges 760
featool/geom2/convert_gobj_polygons.m
CONVERT_GOBJ_POLYGONS Constructs polygons from geometry object . 760
featool/geom2/csg_op.m
CSG_OP Perform CSG operations 760
featool/geom2/csg_polygon_recombination.m
CSG_POLYGON_RECOMBINATION Recombine and tesselate polygons . . 761
featool/geom2/csg_polygon_tesselation.m
CSG_POLYGON_TESSELATION Recombine and tesselate polygons 762
featool/geom2/geom_analyze.m
GEOM_ANALYZE Analyze geometry objects 762
featool/geom2/geom_apply_formula.m
GEOM_APPLY_FORMULA Apply formula to a geometry struct 762
featool/geom2/geom_apply_transformation.m
GEOM_APPLY_TRANSFORMATION Apply transformation to geometry objects 762
featool/geom2/geom_combine_objects.m
GEOM_COMBINE_OBJECTS Combine geometry objects 763
featool/geom2/geom_decompose.m
GEOM_DECOMPOSE Analyze and decompose geometry 763

featool/geom2/gobj_apply_op.m
GOBJ_APPLY_OP Apply operation to geometry objects 764
featool/geom2/gobj_block.m
GOBJ_BLOCK Create block geometry object 764
featool/geom2/gobj_circle.m
GOBJ_CIRCLE Create circle geometry object 765
featool/geom2/gobj_cylinder.m
GOBJ_CYLINDER Create cylinder geometry object 765
featool/geom2/gobj_decompose.m
GOBJ_DECOMPOSE Analyze and decompose geometry object 766
featool/geom2/gobj_ellipse.m
GOBJ_ELLIPSE Create ellipse geometry object 766
featool/geom2/gobj_empty.m
GOBJ_EMPTY Create empty geometry object 767
featool/geom2/gobj_grid.m
GOBJ_GRID Create background grid for geometry objects 767
featool/geom2/gobj_line.m
GOBJ_LINE Create line geometry object 767
featool/geom2/gobj_point.m
GOBJ_POINT Create point geometry object 768
featool/geom2/gobj_polygon.m
GOBJ_POLYGON Create polygon geometry object 768
featool/geom2/gobj_rectangle.m
GOBJ_RECTANGLE Create rectangle geometry object 768
featool/geom2/gobj_sphere.m
GOBJ_SPHERE Create sphere geometry object 769
featool/geom2/match_rows.m
MATCH_ROWS Find rows with identical content in A 769
featool/grid/assign_bdr.m
ASSIGN_BDR Assign boundary numbering according to geometry 769
featool/grid/blockgrid.m
BLOCKGRID Generate 3D regular tensor product hexahedral grid 770
featool/grid/circgrid.m
CIRCGRID Generate 2d quadrilateral grid for a circle 771
featool/grid/constrain_edges.m
CONSTRAIN_EDGES Constrains edges in a triangulation 770

featool/grid/cylgrid.m
CYLGRID Generate 3d cylindrical hexahedral grid 772
featool/grid/delcells.m
DELCELLS Delete cells from grid 772
featool/grid/findbdr.m
FINDBDR Find boundary indices for an expression 773
featool/grid/grid_boundary_nodes.m
GRID_BOUNDARY_NODES Get index to nodes on the boundaries 787
featool/grid/gridadj.m
GRIDADJ Generate grid cell adjacency information 773
featool/grid/gridalign2.m
GRIDALIGN2 Align vertices of quadrilateral cells with interface 774
featool/grid/gridbdr.m
GRIDBDR Generate grid boundary information 774
featool/grid/gridbdr1.m
GRIDBDR1 Generate grid boundary information for one dimensional cells 775
featool/grid/gridbdrx.m
GRIDBDRX Reconstruct interior boundaries 776
featool/grid/gridbfstf.m
GRIDBFSTF Find starting face 776
featool/grid/gridcheck.m
GRIDCHECK Check grid for errors 777
featool/grid/gridconne.m
GRIDCONNE Group connected edges/faces 777
featool/grid/gridedge.m
GRIDEDGE Generate grid cell edge numbering 777
featool/grid/gridextrude.m
GRIDEXTRUDE Extrude 2D grid to 3D 778
featool/grid/gridface.m
GRIDFACE Generate grid cell face numbering 779
featool/grid/gridgen_distmesh.m
GRIDGEN_DISTMESH Generate unstructured grid for geometry objects . . 804
featool/grid/gridgen_gmsh.m
GRIDGEN_GMSH Generate grid with Gmsh 805
featool/grid/gridgen_quad.m
GRIDGEN_QUAD Generate quadrilateral grid for a geometry object 808

featool/grid/gridgen_triangle.m
GRIDGEN_TRIANGLE Generate unstructured 2D grid with Triangle 809
featool/grid/gridmerge.m
GRIDMERGE Merge two grids 780
featool/grid/gridnormals.m
GRIDNORMALS Compute normal vectors 781
featool/grid/gridrefine.m
GRIDREFINE Uniform refinement of a grid 781
featool/grid/gridrefine1.m
GRIDREFINE1 Uniform refinement of a 1D grid 782
featool/grid/gridrefine2.m
GRIDREFINE2 Uniform refinement of a 2D grid 782
featool/grid/gridrefine3.m
GRIDREFINE3 Uniform refinement of a 3D grid 783
featool/grid/gridrevolve.m
GRIDREVOLVE Revolve and extrude a 2D grid 783
featool/grid/gridrotate.m
GRIDROTATE Rotate grid 784
featool/grid/gridscale.m
GRIDSCALE Scale a grid 785
featool/grid/gridsmooth.m
GRIDSMOOTH Grid smoothing 785
featool/grid/gridstat.m
GRIDSTAT Grid statistics 812
featool/grid/gridvert.m
GRIDVERT Generate grid vertex adjacency map 786
featool/grid/hex2tet.m
HEX2TET Convert hexahedra to tetrahedra 792
featool/grid/holegrid.m
HOLEGRID Generate 2d rectangular grid with a circular hole 793
featool/grid/linegrid.m
LINEGRID Generate 1D line grid 794
featool/grid/mesh_boundaries.m
MESH_BOUNDARIES Mesh boundaries 795
featool/grid/mesh_size.m
MESH_SIZE Define mesh sizes 795

featool/grid/project_bdrp.m
PROJECT_BDRP Project boundary points to geometric boundary 797
featool/grid/quad2tri.m
QUAD2TRI Convert quadrilaterals to triangles 797
featool/grid/quadmesh.m
QUADMESH Generate quadrilateral mesh using distance functions 798
featool/grid/reconstruct_bdr.m
ASSIGN_BOUNDARIES Assign and reconstruct boundary struct 799
featool/grid/rectgrid.m
RECTGRID Generate 2D rectangular tensor product quadrilateral grid . . 795
featool/grid/reorient_cells.m
REORIENT_CELLS Reorient cells to corrent orientation 796
featool/grid/ringgrid.m
RINGGRID Generate 2d grid of a ring 796
featool/grid/searchgraph.m
SEARCHGRAPH Find path from two points in edge graph 801
featool/grid/selcells.m
SELCELLS Select cells from expression 799
featool/grid/spheregrid.m
SPHEREGRID Generate 3d hexahedral grid for a sphere 800
featool/grid/tet2hex.m
TET2HEX Convert tetrahedra to hexahedra 801
featool/grid/tri2quad.m
TRI2QUAD Convert triangles to quadrilaterals 801
featool/impexp/geom2geo.m
GEOM2GEO Generate GEO geometry data for use with Gmsh 787
featool/impexp/geom2poly.m
GEOM2POLY Generate poly data for use with Triangle 787
featool/impexp/grid2foam.m
GRID2FOAM Generate/convert grid data to OpenFOAM format 788
featool/impexp/impexp_ascii.m
IMPEXP_ASCII Import or export data in ascii format 788
featool/impexp/impexp_dolfin.m
IMPEXP_DOLFIN Import/export grid in Dolfin XML format 789
featool/impexp/impexp_foam.m
IMPEXP_FOAM Import/export OpenFOAM format 789

featool/impexp/impexp_gid.m
IMPEXP_GID Import/export grid in GID msh format 790
featool/impexp/impexp_gmsh.m
IMPEXP_GMSH Import/export grid in GMSH format 790
featool/impexp/impexp_gmv.m
IMPEXP_GMV Import/export grid in GMV format 791
featool/impexp/impexp_stl.m
IMPEXP_STL Import/export geometry in STL format 791
featool/impexp/impexp_triangle.m
IMPEXP_TRIANGLE Import/export grid data in Triangle format 792
featool/impexp/log_message.m
LOG_MESSAGE Output message to file, terminal, or gui 792
featool/impexp/open_file.m
OPEN_FILE Open file for reading and writing 792
featool/openfoam/openfoam.m
OPENFOAM Exports, solves, and imports an OpenFOAM CFD problem . . 750
featool/openfoam/openfoam_data.m
OPENFOAM_DATA OpenFOAM control and solver parameters 750
featool/openfoam/openfoam_gui.m
OPENFOAM_GUI OpenFOAM gui control dialog box 751
featool/openfoam/openfoam_write.m
OPENFOAM_WRITE Write OpenFOAM dict struct to file 751
featool/physmodes/addphys.m
ADDPHYS Parse and add physics mode to problem struct 815
featool/physmodes/axistressstrain.m
AXISTRESSSTRAIN Axisymmetric stress-strain physics mode 816
featool/physmodes/brinkmaneqns.m
BRINKMANEQNS Brinkman equations physics mode 816
featool/physmodes/conductivemediadc.m
CONDUCTIVEMEDIADC Conductive media DC physics mode 816
featool/physmodes/convectiondiffusion.m
CONVECTIONDIFFUSION Convection and diffusion physics mode definition 817
featool/physmodes/customeqn.m
CUSTOMEQN Custom equation physics mode definition 817

featool/physmodes/darcyslaw.m
DARCYSLAW Darcy's law physics mode . . . 817
featool/physmodes/electrostatics.m
ELECTROSTATICS Electrostatics physics mode . . . 816
featool/physmodes/eulerbeam.m
EULERBEAM 1D Euler-Bernoulli beam physics mode . . . 816
featool/physmodes/heattransfer.m
HEATTRANSFER Heat transfer physics mode definition . . . 817
featool/physmodes/linearelasticity.m
LINEARELASTICITY Linear elasticity physics mode . . . 817
featool/physmodes/list_physmodes.m
LIST_PHYSMODES Lists and orders the available physics modes . . . 817
featool/physmodes/magnetostatics.m
MAGNETOSTATICS Magnetostatics physics mode . . . 816
featool/physmodes/navierstokes.m
NAVIERSTOKES Navier-Stokes equations physics mode . . . 817
featool/physmodes/physmode.m
PHYSMODE Finishes a physics mode and returns the phys struct . . . 817
featool/physmodes/planestrain.m
PLANESTRAIN Plane strain physics mode . . . 818
featool/physmodes/planestress.m
PLANESTRESS Plane stress physics mode . . . 818
featool/physmodes/poisson.m
POISSON Poisson equation physics mode definition . . . 818
featool/post/contour2.m
CONTOUR2 Generate contours on unstructured triangles . . . 821
featool/post/postplot1.m
POSTPLOT1 1D Postprocessing function . . . 821
featool/post/postplot2.m
POSTPLOT2 2D Postprocessing function . . . 822
featool/post/postplot3.m
POSTPLOT3 3D Postprocessing function . . . 822
featool/post/princse.m
PRINCSE Compute principal stresses and strains . . . 823
featool/post/setaxes.m
SETAXES Apply plot axis settings . . . 823

featool/post/slice3.m
SLICE3 Generate a triangular slice grid from tetrahedral grid 824
featool/util/conngrp.m
CONNGRP Group connected sets 827
featool/util/cstrcmp.m
CSTRCMP Compares cells of strings 827
featool/util/deduplicate.m
DEDUPLICATE Remove duplicate rows or columns within tolerance 828
featool/util/dehtmlify.m
DEHTMLIFY Remove html clauses from string 828
featool/util/expandeqn.m
EXPANDEQN Expands an equation string expression 828
featool/util/findparblock.m
FINDPARBLOCK Find parenthesis blocks 829
featool/util/interpl.m
INTERPL Linear interpolation between points 829
featool/util/mergeeqncoefs.m
MERGEEQNCOEFS Merge eqn struct coefficient fields 829
featool/util/parsecoef.m
PARSECOEF Parse coefficients in problem struct 829
featool/util/parseeqn.m
PARSEEQN Parse string equations and return eqn struct 830
featool/util/parseexpr.m
PARSEEXPR Parses a string expression 831
featool/util/parseopt.m
PARSEOPT Parse options 832
featool/util/parsephys.m
PARSEPHYS Check and parse physics modes 832
featool/util/parseprob.m
PARSEPROB Check and parse a problem struct 832
featool/util/parseprob0.m
PARSEPROB0 Low level check and parsing of a problem struct 833
featool/util/recursive_termsplit.m
RECURSIVE_TERMSPLIT Splits a string expression recursively into terms . 833
featool/util/remove_duplicate_rows.m
REMOVE_DUPLICATE_ROWS Removes duplicate rows 834

featool/util/remove_parentheses.m
REMOVE_PARENTHESES Remove outer parentheses from string expression . . . 834
featool/util/rmkdir.m
RMKDIR Recursively create directories . . . 834
featool/util/set_geomlib.m
SET_GEOMLIB Set geomlib version . . . 834
featool/util/stfldcheck.m
STFLDCHECK Traverse the struct ST0 and checks if all fields in C exist . . . 835
featool/util/strconv.m
STRCONV Convert string into numeric and string data . . . 835
featool/util/stringify.m
STRINGIFY Convert input to string . . . 835
featool/util/symvarbe.m
SYMVARBE Determine the symbolic variables in an expression . . . 836
featool/util/termsplit.m
TERMSPLIT Splits string into terms . . . 836
featool/util/timerfun.m
TIMERFUN Create single timer function . . . 836
featool/util/urlget.m
URLGET Get/fetch file from url . . . 837
featool/util/uunique.m
UUNIQUE Unsorted set unique . . . 837

11 Function Reference

11.1 addpaths

ADDPATHS Add/remove FEATool directories to/from working paths.

[C_PATHS] =

ADDPATHS(I_ADD) Adds/removes FEATool directories to/from working paths. The I_ADD flag >1 (default) indicates adding paths, and <0 removing paths. Optionally returns the paths in C_PATHS where the first column consists of all paths relative to the FEATool root installation directory, and the second with corresponding absolute paths.

11.2 core/asmgroups

ASMGROUPS Determine cell indices for assembly subdomian/cell groups.

[CGROUPS, N_MAX] =

ASMGROUPS(N_MAX_CELLS, SIND, C) Determines the cell indices for the assembly cell groups with no more than N_MAX_CELLS in each group. SIND is the integer vector indexing which subdomian each cell belongs to. Returns the cell indices for each group in CGROUPS and the maximum number of cells in all groups in N_MAX. If the optional third input argument C (grid cell connectivties) is given, then groups will be split so that no cells in each group are connected through the vertices.

See also

assemblea, assemblef

11.3 core/asmrowcolptr

ASMROWCOLPTR Compute row and column pointers for sparse matrix assembly.

[VROWINDS, VROWCOLS, N_ROWS, N_COLS] =

ASMROWCOLPTR(C_SFUN, P, C, A, N_MAX_CELLS, ADOFMAPTRIAL, ADOFMAPTEST) Determines the row and column pointer arrays for a bilinear form.

```
Input        Value/[Size]           Description
-------------------------------------------------------------------------------
c_sfun       strings [2]            Shape functions used for the trial and test
                                    function spaces respectively
p            [n_sdim,n_p]           Array with grid point coordinates
c            [n_vert,n_cells]       Array with cell connectivities, for each
                                    cell (column) points to vertices in p
a            [n_vert,n_cells]       Array with cell adjacency info, for each
cgroups      [n_groups]             Cell array of cell group indices
aDofMapTrial [n_ldof1,n_cells]      Precomputed local to global dof map (trial)
aDofMapTest  [n_ldof2,n_cells]      Precomputed local to global dof map (test)
                                                                              .
Output       Value/[Size]           Description
-------------------------------------------------------------------------------
vRowInds     [n_A]                  Row pointers to matrix entries
vColInds     [n_A]                  Column pointers to matrix entries
n_rows       scalar                 Number of rows in matrix
n_cols       scalar                 Number of columns in matrix
```

See also

assemblea, assemblef

11.4 core/assemblea

ASSEMBLEA Finite element assembly of bilinear forms.

[VROWINDS, VROWCOLS, VAVALS, N_ROWS, N_COLS] =

ASSEMBLEA(FORM, C_SFUN, C_COEF, ICUB, P, C, A, SIND, N_CMAX, N_OFF, PROB, I_HRZ) Assembles a matrix representation of a finite element bilinear form.

Each column in FORM specify one bilinear additive term to compute. The first row corresponds to the trial function space, and second test space. The FORM entries denote either function values or derivatives to evaluate.

C_SFUN indicates which shape functions to use for the trial and test spaces.

C_COEF specifies the coefficients to evaluate for each term (column) in FORM.

The computed matrix entries are returned in VAVALS where VROWINDS and VROWCOLS are corresponding row and column pointers. N_ROWS and N_COLS give the number of rows in the matrix. A sparse representation of the matrix can subsequently be constructed by calling SPARSE

```
A = sparse( vRowInds, vColInds, vAvals, n_rows, n_cols );
```

on the output of ASSEMLBLEA.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
form        [2,n_forms]            Forms to compute, first row indicates trial
                                   function space (second test function space)
                                    1     = function value
                                    2/3/4 = x/y/z -derivatives
c_sfun      strings {2}            Shape functions used for the trial and
                                   test function spaces
c_coef      {n_forms}              Coefficients for each bilinear form
i_cub       scalar                 Numerical integration rule
p           [n_sdim,n_p]           Array with grid point coordinates
c           [n_vert,n_cells]       Array with cell connectivities
a           [n_vert,n_cells]       Array with cell adjacency information
sind        [n_cells]              (optional) Subdomain indices
n_cmax      scalar                 (optional) Max number of cells to assemble
                                   for at once (to limit memory consumption)
n_off       [2]                    (optional) Row/column pointer offset
prob        struct                 (optional) FEATool problem struct, used
                                   for dofmapping and evaluating coefficients
i_hrz       scalar {0}             Apply HRZ diagonal (mass) lumping
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vRowInds    [n_A]                  Row pointers to matrix entries
vColInds    [n_A]                  Column pointers to matrix entries
vAvals      [n_A]                  Values of the assembled matrix entries
n_rows      scalar                 Number of rows in matrix
n_cols      scalar                 Number of columns in matrix
```

See also

assembleprob, asmrowcolptr, mapdofbdr

11.5 core/assemblef

ASSEMBLEF Finite element assembly of linear form (rhs/load vector).

[F] =

ASSEMBLEF(FORM, C_SFUN, C_COEF, ICUB, P, C, A, SIND, N_CMAX, PROB) Assembles a finite element linear form (right hand side/load vector). Each column in FORM specify one linear additive term to compute. The FORM entries denote either function values or derivatives to evaluate. C_COEF specifies the coefficients to evaluate for each term in FORM. C_SFUN indicates which shape functions to use for the evaluation.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
form        [n_forms]              Linear forms to compute and add
             1                      - function value
             2,3,4                  - x,y,z -derivatives
c_sfun      string                 Shape functions used for the trial space
c_coef      [n_forms]              Coefficients to multiply/scale each form
i_cub       scalar                 Numerical integration rule
p           [n_sdim,n_p]           Array with grid point coordinates
c           [n_vert,n_cells]       Array with cell connectivities, for each
                                   cell (column) points to vertices in p
a           [n_vert,n_cells]       Array with cell adjacency info, for each
                                   cell (column) points to cell neighbours
sind        [n_cells]              Subdomain indices
n_cmax      scalar                 (optional) Max number of cells to assemble
                                   for at once (to limit memory consumption)
prob        struct                 Finite element problem struct, (optionally)
                                   used for evaluating the coefficients
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
f    [n_f]                         Right hand side/load vector
```

See also

assembleprob, mapdofbdr

11.6 core/assemblej

ASSEMBLEJ Finite element assembly of Jacobian forms.

[VROWINDS, VROWCOLS, VJVALS, N_ROWS, N_COLS] =

ASSEMBLEJ(AFORM, FFORM, C_SFUN, C_ACOEF, C_FCOEF, ICUB, P, C, A, IND_S, PROB, OFFSET, IPERT, DPERT, N_CMAX)

Assembles a numerical Jacobian matrix representation of a finite element bilinear and linear forms.

IPERT indicates to perturb the solution vector (=0) with DPERT (default 1e-8) calculating the Jacobian J = dK/du∗U - f, where each column J_i is given as

```
J_i = ( (K(U_i+DPERT)*(U_i+DPERT)-f(U_i+DPERT)) - (K(U)*U-f(U)) )/DPERT
```

Alternatively, if IPERT is equal to an integer (1, 2 or 3) the Jacobian with respect to space dimension/coordinate IPERT will be computed as

```
J_i = ( (K(U,x+DPERT)*(U,x+DPERT)-f(U,x+DPERT)) - (K(U,x)*U-f(U,x)) )/DPERT
```

The linearization point/solution vector U needs to be given and stored in the prob.sol.u field. If it is not present a zero solution will be used.

Each column in AFORM and FFORM specify bilinear and linear additive terms to compute. In AFORM the first row corresponds to the trial function space, and second test space. The form entries denote either function values or derivatives to evaluate.

C_SFUN indicates which shape functions to use for the trial and test function spaces. Optionally, a third entry is required for the trial function space when assembling off-diagonal (target) blocks for coupled problems. If only one or two C_SFUN entries is provided, the first entry will be used for target trial function space.

For off-diagonal blocks the dof OFFSET (dof_offset_row, dof_offset_column) is used to find and perturb the target block solution vector.

C_ACOEF and C_FCOEF specifies the coefficients to evaluate for the forms.

The computed matrix entries are returned in VJVALS where VROWINDS and VROWCOLS are

corresponding row and column pointers. N_ROWS and N_COLS give the number of rows in the matrix. J sparse representation of the matrix can subsequently be constructed by calling SPARSE

```
J = sparse( vRowInds, vColInds, vJvals, n_rows, n_cols );
```

on the output of ASSEMLBLEJ.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
aform       [2,n_aforms]           Bilinear forms to compute, first row indicates
                                   trial function space (second test function space)
                                    1     = function value
                                    2/3/4 = x/y/z -derivatives
fform       [n_fforms]             Linear forms to compute
c_sfun      strings {2/3}          Shape functions used for the assmebly trial,
                                   assembly test, and target trial function spaces
c_acoef     {n_aforms}             Coefficients for each bilinear form
c_fcoef     {n_fforms}             Coefficients for each linear form
i_cub       scalar                 Numerical integration rule
p           [n_sdim,n_p]           Array with grid point coordinates
c           [n_vert,n_cells]       Array with cell connectivities
a           [n_vert,n_cells]       Array with cell adjacency information
ind_s       [n_cells]              (optional) Subdomain indices
prob        struct                 (optional) FEATool problem struct, used
offset      [2]                    (optional) Offset for finding start in solution
                                   for matrix coef/vector product dof perturbation
ipert       scalar/{0/i_sdim}      0-Solution perturbation, >0 Coordinate dir. pert
dpert       scalar                 (optional) Perturbation parameter (default 1e-8)
n_cmax      scalar                 Max number of cells per group to assemble
                                   for at once (to limit memory consumption)
                                                                               .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vRowInds    [n_J]                  Row pointers to matrix entries
vColInds    [n_J]                  Column pointers to matrix entries
vJvals      [n_J]                  Values of the assembled matrix entries
n_rows      scalar                 Number of rows in matrix
n_cols      scalar                 Number of columns in matrix
```

See also

assemjac, asmrowcolptr, mapdofbdr

11.7 core/assemblenjac

ASSEMBLENJAC Assembly of defect and Newton Jacobian.

[D, J] =

ASSEMBLENJAC(U, PROB, VARARGIN) Evaluates defect with solution vector U and optionally assembles the nonlinear Jacobian matrix. Accepts optional propery value pairs in VARARGIN.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
u           vector [n_u,1]         Solution vector for linearization point
prob        struct                 Problem definition struct
jac         scalar/struct {0}      Jacobian computation type
                                     . - Direct assembly using symbolic jac.form
                                         and jac.coef struct contents
                                    0 - Assembly using numerical differentiation
                                   >0 - Directional Jacobian (jac = i_sdim dir.)
                                   <0 - Numeric Mat-Prod Jacobian construction
du          scalar {1e-8}          Magnitude of numerical differentiation shift
icub        scalar {2}             Numerical integration rule
n_cmax      scalar {50000}         Max number of cells to assemble
                                   for at once (to limit memory consumption)
f_sparse    scalar {1}             Return Matlab sparse matrix format
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
d           vector [n_u,1]         Defect vector, d := f(u) - A(u)*u
J           sparse [n_u,n_u]       Assembled Jacobian matrix (optional output)
```

See also

assemmat, assemjac, assemblef

11.8 core/assembleprob

ASSEMBLEPROB Assembly of system matrix and right hand side/load vector.

[M, A, F, T_M, T_A, T_F, T_SP] =

ASSEMBLEPROB(PROB, VARARGIN) Calls the k assembly routines to compute and assemble a monolithic sparse system matrix and a right hand side/load vector as specified in the PROB struct. Accepts optional propery value pairs in VARARGIN.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
prob        struct                 Problem definition struct
icub        scalar {2}             Numerical integration rule
imass       scalar {1}             Mass matrix lumping:  1 = Full mass matrix
                                   2 = row sum lumping,  3 = diagonal lumping
                                   4 = HRZ diagonal lumping
n_cmax      scalar {50000}         Max number of cells to assemble
                                   for at once (to limit memory consumption)
f_m         scalar {0}             Assembly flag for mass matrix
f_a         scalar {0}             Assembly flag for system matrix
f_f         scalar {0}             Assembly flag for load vector
f_sparse    scalar {0}             Return Matlab sparse matrix format
solcomp     {all dvars/subd}       Dependent variables/subdomains to assemble for
                                                                               .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
M           sparse [n_M]           Assembled mass matrix
A           sparse [n_A]           Assembled system matrix
f           [n_A,1]                Assembled rhs/load vector
t_m         scalar                 Time spent assembling mass matrix
t_a         scalar                 Time spent assembling system matrix
t_f         scalar                 Time spent assembling rhs/load vector
t_sp        scalar                 Time for sparse matrix conversion
```

See also

assemblea, assemblef, assemmat

11.9 core/assemjac

ASSEMJAC Assemble monolithic Jacobian matrix.

[J, T_J, T_SP] =

ASSEMJAC(PROB, U, METHOD, SOLCOMP, DPERT, I_CUB, F_SPARSE)

Called from assemblenjac to assemble a monolithic Jacobian matrix J.

The METHOD flag indicates computing the Jacobian J = d(K(U)∗U-f(U))/dU with respect to the solution and linearization point U (METHOD=0), where each column J_i is computed as

```
J_i = ( (K(U_i+DPERT)*(U_i+DPERT)-f(U_i+DPERT)) - (K(U)*U-f(U)) )/DPERT
```

or alternatively a coordinate direction (METHOD=j), Jj = d(K(U,xj)∗U-f(U,xj))/dxj where

```
J_i = ( (K(U,xj+DPERT)*(U,xj+DPERT)-f(U,xj+DPERT)) - (K(U,x)*U-f(U,xj)) )/DPERT
```

The optional input argument SOLCOMP can for (METHOD=0) be specified as a 2 x n (cell) array indicating which row and column (dependent variable) combinations should be computed and returned. The first column corresponds to the row and second column for the blocks. SOLCOMP can also equivalently be prescribed as a cell array of pairwise dependent variable name strings. For (METHOD=1/2/3) SOLCOMP can be a single vector to specify the blocks/dependent variable to compute and return. If SOLCOMP is not prescribed or empty all blocks and dependent variable will be computed.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
prob        struct                 Problem definition struct
u           vector [n_u,1]         Solution vector/linearization point, size(n_u,1)
method      scalar {0/i_sdim}      Jacobian computation type
                                       0 - Jacobian with respect to solution
                                   1/2/3 - Jacobian with respect to direction
solcomp     {all dvars/subd}       Dependent variables/subdomains to assemble for
dpert       scalar/{1e-8}          Perturbation parameter
icub        scalar/{2}             Numerical integration rule
f_sparse    scalar/{true}          Return sparse matrix format
                                                                              .
Output      Value                  Description
-------------------------------------------------------------------------------
J           sparse/triplet         Assembled Jacobian matrix (size(n_u,n_u))
t_j         scalar                 Time spent assembling Jacobian matrix
t_sp        scalar                 Time for sparse matrix conversion
```

See also

assemblenjac, assemblej

11.10 core/assemmat

ASSEMMAT Assemble monolithic matrix.

[A, T_A, T_SP] =

ASSEMMAT(PROB, S_A, I_CUB, N_CMAX, F_SPARSE, I_HRZ, SOLCOMP) Called from assembleprob to assemble a monolithic matrix.

See also

assemblea

11.11 core/bdrneu

BDRNEU Compute Neumann (flux) boundary condition vector.

[F, T] =

BDRNEU(PROB, I_CUB) Computes a (flux) load vector with Neumann boundary condition contributinos to the problem specified in the problem struct PROB.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
prob        struct                 Finite element problem struct
i_cub       scalar                 Numerical integration rule
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
f           (neq,1)                Neumann boundary condition load vector
t           scalar                 Time spent in function
```

11.12 core/bdrneu_lin

BDRNEU_LIN Linearize Neumann boundary contributions.

[C_LCOEF, C_COEF] =

BDRNEU_LIN(C_COEF, C_DVAR, N_BDR) Attempts to find terms in the boundary coefficients C_COEF with simple linear products of the dependent variables in C_DVAR and linearize them. Returns the linearized coefficients in C_LCOEF and the remaining coefficents in C_COEF (N_BDR specifies the number of boundaries present).

11.13 core/bdrsetd

BDRSETD Set Dirichlet boundary conditions.

[AMAT, F, T] =

BDRSETD(AMAT, F, PROB, ISYMM, SET_NULL, SOLCOMP) Sets Dirichlet boundary conditions in the global matrix AMAT and right hand side/load vector F with the information in the finite element problem struct PROB. The optional flag ISYMM symmetrizes the boundary conditions. SET_NULL prescibes zeros instead of the expression specified in the prob.bdr.d field.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
amat        (n_a,n_a)              System matrix (sparse or triplet format)
f           (neq,1)                Right hand side/load vector
prob        struct                 Finite element problem struct
isymm       scalar/{0}             Symmetrize BCs if applicable
set_null    scalar/{0}             Set zeros in f vector
solcomp     {all dvars/subd}       Dependent variables/subdomains to set BCs for
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
amat        (n_a,n_a)              Modified system matrix
f           (neq,1)                Modified right hand side/load vector
t           scalar                 Time spent in function
```

11.14 core/bdrcpl

BDRCPL Set coupled Dirichlet boundary conditions.

[A, F, PROB] =

BDRCPL(A, F, PROB, I_DVAR, J_BDR, SOLVE_STEP) Couples I_DVAR and J_DVAR on the shared boundary J_BDR. J_DVAR is defined in the persistent variable ARGS by pre-calling the function with the single argument
[I_DVAR J_BDR J_DVAR FAC_I FAC_J], where FAC_I and FAC_J are the relative scaling of the coupled variables (default 1) so that FAC_I∗I_DVAR = FAC_J∗J_DVAR is satisfied on J_BDR.

See also

solve_hook

11.15 core/bdrslip

BDRDLIP Set slip boundary conditions.

[A, F, PROB] =

BDRSLIP(A, F, PROB, I_DVAR, J_BDR, SOLVE_STEP) Sets slip boundary conditions in the global matrix A and right hand side/ load vector F with the information in the finite element problem struct PROB. I_DVAR inidicates dependent variable number (should correspond to x-velocity) and J_BDR the boundary number. SOLVE_STEP indicates if function call is before the linear solver call (=1) or after (=2).

See also

solve_hook

11.16 core/cellvol

CELLVOL Computes cell volume for cells.

[V] =

CELLVOL(P, C) Computes the cell volumes for the cells in C with vertices in P.

Input	Value/(Size)	Description
p	(n_sdim,n_p)	Array of cell vertex coordinates
c	(n_pcell,n_c)	Index array for cells to evaluate

Output	Value/(Size)	Description
V	(n_c)	Cell volumes

11.17 core/coordr2l

COORDR2L Convert real to local coordinates.

[XI, CIND] =

COORDR2L(XP, P, C) Converts the real coordinates in XP to local coordinates XI and also returns indices to the corresponding cells in CIND. P and C is a valid column ordered grid cell triangulation.

11.18 core/coordtf

COORDTF Transforms (interpolates) coordinates from local to real coordinates.

[X] =

COORDTF(I_SDIM, XI, P, C) Computes the transformation (interpolation) of the local coordinate XI to real coordinates for all cells indexed in C. P is an array of cell vertex coordinates and I_SDIM specifies which coordinate direction to compute.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
i_sdim      1-3                    Coordinate direction to evaluate
xi          (1-4)                  Local cell coordinates to evaluate
p           (n_sdim,n_p)           Array of cell vertex coordinates
c           (n_pcell,n_c)          Index array for cells to evaluate
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
x           (n_c)                  Evaluated coordinate values
```

11.19 core/quadrule

QUADRULE Calculate numerical quadrature/integration rules.

[X, W, N] =

QUADRULE(N0, N_SDIM, N_PCELL) Computes numerical quadrature integration rules for the appropriate grid cell type determined by the space dimension N_SDIM and the number of points/vertices per cell N_PCELL. N0 specifies the desired order and degree of the rule. Returns evaluation coordinates in X (Barycentric for lines and simplices, and reference coordinates for quadilaterals and hexahedra), weights in W, and the total number of integration points in N.

Tabulated Gauss quadrature rules are used for triangles and tetrahedra with N0 <= 5, while for the rest open Gauss-Legendre multidimensional (and collapsed) quadrature is used. Note that the 3rd degree/order triangle and tetrahedral rule include negative weights.

See also

quadruleb

11.20 core/quadruleb

QUADRULEB Calculates numerical integration rules for cell boundaries.

[X, W, N] =

QUADRULEB(N0, N_SDIM, N_PCELL, ADD_BDR) Computes numerical integration rules for boundary edges (2D) or faces (3D) and corresponding cell types. Returns the evaluation coordinates in X, weights in W, and the total number of integration points in N. If the optional flag ADD_BDR is set then cell vertex (and edge in 3D) points are added explicitly with zero weights.

See also

quadrule

11.21 core/quadefc

QUADEFC Get local quadrature coordinates on edges/faces.

[XIE] =

QUADEFC(XI, N_SDIM, I_EF, N_PCELL) Determines the local evaluation coordinates of point XI, given in local reference coordinates on edge or face I_EF for a N_SDIM dimensional cell with N_PCELL vertices.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
xi          array                  Local reference coordinates on edge/face
n_sdim      scalar                 Number of space dimensions
i_ef        scalar                 Local edge/face number
n_pcell     scalar                 Number of vertices per cell
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
xie         array                  Local quadrature coordinates
```

11.22 core/evaldvar

EVALDVAR Evaluates a dependent variable in a point on a group of cells.

[VDVARVAL] =

EVALDVAR(SFUN, I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, C, U) Evaluates a dependent variable in point XI on the cells given in C. SFUN specifies which shape function to use with the corresponding solution vector U.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
sfun        string                 Shape function string
i_eval      scalar                 Evaluation flag
n_sdim      scalar                 Number of space dimensions
n_vert      scalar                 Number of vertices per cell
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
c           [n_ldof,n_cells]       Dof connectivity map (for each cell)
u           [n_u]                  Solution vector
                                                                            .
Output      Value/(Size)           Description
-----------------------------------------------------------------------------
vDVarVal    [n_cells]              Evaluated values
```

See also

evalexpr, evalsfun

11.23 core/evalexpr0

EVALEXPR0 Evaluates an expression in a point on a group of cells.

[VEVAL] =

EVALEXPR0(S_EXPR, XI, IND_S, IND_C, IND_E, PROB, AJAC) Given a valid PROB struct evaluates the expression S_EXPR on cells with index IND_C at point XI in local coordinates.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
s_expr      string                 String expression to evaluate
xi          [n_sdim(+1),1]         Local coordinates of evaluation point
ind_s       scalar                 Subdomain to evaluate coefficients
ind_c       [n_c]                  Index vector to cells to evaluate
ind_e       [n_c]                  Index vector to edges to evaluate
```

```
prob        struct                Problem definition struct
aJac        (n_c,:)               Optional array for Jac. inverse and determinant
                                                                                .
Output      Value/(Size)          Description
--------------------------------------------------------------------------------
vEval       [n_c]                 Output vector of evaluated values
```

See also

evaldvar, evalsfun

11.24 core/evalexprc

EVALEXPRC Evaluates an expression in cells.

[AEVAL] =

EVALEXPRC(S_EXPR, PROB, INDC, SOLNUM) Evaluates the expression S_EXPR in the cells specified by INDC. PROB is a valid finite element problem struct and the optional argument SOLNUM specifies the solution number to use.

```
Input       Value/[Size]          Description
--------------------------------------------------------------------------------
s_expr      string                String expression to evaluate
ind_c       [n_c]                 Index vector to cells to evaluate
prob        struct                Problem definition struct
solnum      scalar                Solution number/time to use in evaluation
                                                                                .
Output      Value/(Size)          Description
--------------------------------------------------------------------------------
aEval       [n_pcell,n_c]         Output array of evaluated values
```

11.25 core/evalexprp

EVALEXPRP Evaluates an expression in the grid points.

[VEVAL] =

EVALEXPRP(S_EXPR, PROB, SOLNUM) Evaluates the expression S_EXPR in the points specified in PROB.GRID.P (in global x/y/z-coordinates). PROB is a valid finite element problem struct.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
s_expr      string                 String expression to evaluate
prob        struct                 Problem definition struct
solnum      scalar                 Solution number/time to use in evaluation
                                                                            .
Output      Value/(Size)           Description
-----------------------------------------------------------------------------
vEval       [n_p,1]                Output vector of evaluated values
```

11.26 core/evalinit

EVALINIT Calculates initial solution vector from given expressions.

[U0] =

EVALINIT(PROB) Given a valid PROB struct evaluates the inital value expressions and returns a solution vector U0.

See also

evalexpr

11.27 core/evalsfun

EVALSFUN Shape function evaluation driver subroutine.

[VBASE, NLDOF, XLDOF, SFUN] =

EVALSFUN(SFUN, I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates local basis function I_DOF in point XI for shape function SFUN on all cells corresponding to AINVJAC. Returns the evaluated values in VBASE and the local number of degrees of freedom on vertices, edges, faces, and cell interiors in NLDOF and XLDOF as well as the low level function name in SFUN.

See also

evaldvar, evalexpr

11.28 core/lcoorde

LCOORDE Get local coordinates on a cell edge/face.

[XI] =

LCOORDE(I_E, DT, N_V, N_SDIM) Computes local coordinates on edge/face I_E on a N_SDIM dimensional cell with N_V vertices. DT specifies the relative distance(s) of the evaluation point(s) from corner/edge I_E.

11.29 core/mapdofbdr

MAPDOFBDR Create degree of freedom and boundary maps.

[N_GDOF, ADOFMAP, ABDRMAP] =

MAPDOFBDR(VARARGIN) Computes a connectivity map to the global degrees of freedom on each cell, and optionally computes a boundary index vector for all degrees of freedom. Input parameters can either be old type call (P,C,A,B,NLDOF) or (C,B,NLDOF,XI) where the local dof coordinates XI is optional.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
p           (2,n_p)                Array with grid point coordinates
c           (n_vc,n_c)             Array with cell connectivities, points
                                   to vertices in p for each cell
a           (n_cf,n_c)             Cell adjacency information, points to
                                   adjacent cells for each edge. If the edge
                                   is on a boundary the a entry is zero
b           (4-6,n_cb)             Boundary indicies for degress of freedom
nLDof       (n_bdgroups,4)         Number of local degrees of freedom on vertices
                                   edges, and cell interiors for boundary groups
xi          (2-4,sum(nLDof))       Coordinates of local dofs (optional).
                                                                               .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
n_gdof      (scalar)               Total number of degrees of freedom
aDofMap     (n_ldof,n_c)           Connectivity map giving degree of freedom
                                   numbers for each local dof on all cells
aBdrMap     (5+n_sdim,n_bdof)      Boundary dof map. First row is the cell number,
                                   followed by edge/face, boundary, global and
```

```
                                        local dof numbers, and local coordinates
                                        on edges/faces.
```

11.30 core/re2locquad

RE2LOCQUAD Convert quadrialteral real to local coordinates.

[XI] =

RE2LOCQUAD(P, DX, DY) Converts the real coordinates in P to local coordinates XI for quadrilateral cells defined by the vertices given in (DX,DY). P is a 2 by n_p array, and DX/DY 4 by n_p arrays for vectorized evaluation.

11.31 core/solve_hook

SOLVE_HOOK Custom solver hook calling function.

[A, F, PROB, C_FCN] =

SOLVE_HOOK(A, F, PROB, SOLVE_STEP) Special solver hook function. Allows for special functions to be called before and after the linear solver to modify the global matrix A and right hand side/load vector F with the information in the finite element problem struct PROB. SOLVE_STEP indicates if the function call is before the linear solver call (=1) or after (=2). The called solver hook function names are returned in the cell array C_FCN.

The solve hook function can be defined as a Neumann boundary condition string (fea.bdr.n entry) as

```
fea.bdr.n{i_dvar}{j_bdr} = <tt>solve_hook_fcn</tt>;
```

The function name is specified after the "solve_hook_" string prefix (here called "fcn), where the solve hook function call format is

```
fcn(  A, f, fea, i_dvar, j_bdr, solve_step )
```

For example, the Navier-Stokes slip boundary conditions are implemented using the solve hook functionality by prescribing

```
fea.bdr.n{dvar_u}{j_bdr} = <tt>solve_hook_bdrslip</tt>;
```

for the Neumann BC component in the x-direction. This will cause the solve hook to trigger calling the "bdrslip" function before and after the linear solver.

See also

bdrslip, bdrcpl

11.32 core/pntset

PNTSET Set Dirichlet point constraints.

[F, INDROW, AMAT, T] =

PNTSET(PROB, F, AMAT, ISYMM, SET_NULL) Sets Dirichlet point constraints conditions in the right hand side load vector F and global matrix AMAT with the information in the finite element problem struct PROB.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
prob        struct                 Finite element problem struct
f           (neq,1)                Right hand side/load vector
amat        (n_a,n_a)              System matrix (sparse or triplet format)
isymm       scalar/{0}             Symmetrize BCs if applicable.
set_null    scalar/{0}             Set zeros in f vector.
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
f           (neq,1)                Modified right hand side/load vector
indrow      (neq,1)                Index to rows (dofs) in rhs which were set
amat        (n_a,n_a)              Modified system matrix
t           scalar                 Time spent in function
```

11.33 core/pntsetf

PNTSETF Assemble point source terms.

[F, INDROW, T] =

PNTSETF(PROB, I_CUB, F) Assembles point sources to the right hand side load vector F with the information in the finite element problem struct PROB.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
prob        struct                 Finite element problem struct
i_cub       scalar                 Numerical integration rule
f           (neq,1)                Right hand side/load vector
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
f           (neq,1)                Modified right hand side/load vector
indrow      (neq,1)                Index to rows (dofs) in rhs which were set
t           scalar                 Time spent in function
```

11.34 core/subcoef

SUBCOEF Substitute coefficients from cell array.

[S_EXPR] =

SUBCOEF(S_EXPR, CCOEF, I_SUBCELL) Recursively parse the string exptression S_EXPR for matching coefficients in CCOEF. The first column in CCOEF corresponds to the coefficient names and the last to their corresponding expressions. If a coefficient expression consists of a cell array then entry number I_SUBCELL will be used (default 1).

11.35 core/tfjac

TFJAC Computes transformation Jacobian and determinant.

[AJTMP, AJAC] =

TFJAC(I_FLAG, P, C, AJTMP, XI, AJAC) Computes the inverse of the transformation Jacobian and corresponding determinant in point XI (in local coordinates). The input and output array AJTMP saves and reuses factors that are independent of the evaluation point XI.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
i_flag      (scalar)               Evaluation flag (0=Help array, 1=All, 2=Jac.)
p           (n_sdim,n_p)           Array of cell vertex coordinates
c           (n_pcell,n_c)          Index array for cells to evaluate
```

```
aJTmp       (n_c,:)               Temporary help array
xi          (n_sdim(+1))          Local cell coordinates to evaluate
aJac        (n_c,:)               Array for Jacobian inverse and determinant
store_aJTmp (scalar)              Flag to store aJTmp as a persistent variable

Output      Value/(Size)          Description
                                                                                .
--------------------------------------------------------------------------------
aJTmp       (n_c,:)               Temporary help array
aJac        (n_c,:)               Array for Jacobian inverse and determinant
```

See also

tfjacline, tfjactri, tfjacquad, tfjactet, tfjachex

11.36 core/tfjachex

TFJACHEX Computes transformation Jacobian inverse and determinant for hexahedrals.

[AJTMP, AJAC] =

TFJACHEX(I_FLAG, P, C, AJTMP, XI, AJAC) Computes the inverse of the trilinear transformation Jacobian and corresponding determinant in point XI on hexahedral cells. The optional input and output array AJTMP saves and reuses factors that are independent of the evaluation point.

```
Input       Value/(Size)          Description
--------------------------------------------------------------------------------
i_flag      (scalar)              Evaluation flag (0=Help array, 1=All, 2=Jac.)
p           (3,n_p)               Cell vertex coordinates
c           (8,n_c)               Cell connectivities
aJTmp       (n_c,21)              Temporary help array
xi          (3)                   Local cell coordinates to evaluate in
aJac        (n_c,10)              Array for Jacobian inverse and determinant
store_aJTmp (scalar)              Flag to store aJTmp as a persistent variable
                                                                                .
Output      Value/(Size)          Description
--------------------------------------------------------------------------------
aJTmp       (n_c,21)              Temporary help array
aJac        (n_c,10)              Array for Jacobian inverse and determinant
```

See also

tfjac, tfjacline, tfjactri, tfjacquad, tfjactet

11.37 core/tfjacline

TFJACLINE Computes transformation Jacobian inverse and determinant for lines.

[AJTMP, AJAC] =

TFJACLINE(I_FLAG, P, C, AJTMP, XI, AJAC) Computes the inverse of the linear transformation Jacobian and corresponding determinant in point XI on lines. The optional input and output array AJTMP saves and reuses factors that are independent of the evaluation point.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
i_flag      (scalar)               Evaluation flag (0=Help array, 1=All, 2=Jac.)
p           (1,n_p)                Cell vertex coordinates
c           (2,n_c)                Cell connectivities
aJTmp       (n_c,3)                Temporary help array
xi          (2)                    Barycentric cell coordinates to evaluate in
aJac        (n_c,3)                Array for Jacobian inverse and determinant
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
aJTmp       (n_c,3)                Temporary help array
aJac        (n_c,3)                Array for Jacobian inverse and determinant
```

See also

tfjac, tfjactri, tfjacquad, tfjactet, tfjachex

11.38 core/tfjacquad

TFJACQUAD Computes transformation Jacobian and determinant for quadrilaterals.

[AJTMP, AJAC] =

TFJACQUAD(I_FLAG, P, C, AJTMP, XI, AJAC) Computes the inverse of the linear transformation Jacobian and corresponding determinant in point XI on quadrilateral cells. The optional input and output array AJTMP saves and reuses factors that are independent of the evaluation point.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
i_flag      (scalar)               Evaluation flag (0=Help array, 1=All, 2=Jac.)
p           (2,n_p)                Cell vertex coordinates
c           (4,n_c)                Cell connectivities
aJTmp       (n_c,4)                Temporary help array
xi          (2)                    Local cell coordinates to evaluate in
aJac        (n_c,5)                Array for Jacobian inverse and determinant
store_aJTmp (scalar)               Flag to store aJTmp as a persistent variable
                                                                               .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
aJTmp       (n_c,4)                Temporary help array
aJac        (n_c,5)                Array for Jacobian inverse and determinant
```

See also

tfjac, tfjacline, tfjactri, tfjactet, tfjachex

11.39 core/tfjactet

TFJACTET Computes transformation Jacobian inverse and determinant for tetraheda.

[AJTMP, AJAC] =

TFJACTET(I_FLAG, P, C, AJTMP, XI, AJAC) Computes the inverse of the linear transformation Jacobian and corresponding determinant in point XI on tetrahedral cells. The optional input and output array AJTMP saves and reuses factors that are independent of the evaluation point.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
i_flag      (scalar)               Evaluation flag (0=Help array, 1=All, 2=Jac.)
p           (3,n_p)                Cell vertex coordinates
c           (4,n_c)                Cell connectivities
aJTmp       (n_c,13)               Temporary help array
xi          (4)                    Barycentric cell coordinates to evaluate in
aJac        (n_c,13)               Array for Jacobian inverse and determinant
                                                                               .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
aJTmp       (n_c,13)               Temporary help array
aJac        (n_c,13)               Array for Jacobian inverse and determinant
```

See also

tfjac, tfjacline, tfjactri, tfjacquad, tfjachex

11.40 core/tfjactri

TFJACTRI Computes transformation Jacobian inverse and determinant for triangles.

[AJTMP, AJAC] =

TFJACTRI(I_FLAG, P, C, AJTMP, XI, AJAC) Computes the inverse of the linear transformation Jacobian and corresponding determinant in point XI on triangular cells. The optional input and output array AJTMP saves and reuses factors that are independent of the evaluation point.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
i_flag      (scalar)               Evaluation flag (0=Help array, 1=All, 2=Jac.)
p           (2,n_p)                Cell vertex coordinates
c           (3,n_c)                Cell connectivities
aJTmp       (n_c,7)                Temporary help array
xi          (3)                    Barycentric cell coordinates to evaluate in
aJac        (n_c,7)                Array for Jacobian inverse and determinant
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
aJTmp       (n_c,7)                Temporary help array
aJac        (n_c,7)                Array for Jacobian inverse and determinant
```

See also

tfjac, tfjacline, tfjacquad, tfjactet, tfjachex

11.41 ellib/sf_disc0

SF_DISC0 Constant discontinuous shape function.

[VBASE, NLDOF, XLDOF, SFUN] =

SF_DISC0(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates discontinuous constant shape functions.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
```

```
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_disc1

Code listing

```
nLDof = [0 0 0 1];
if( n_vert==(n_sdim+1) )
  xLDof = ones(n_sdim+1,1)/(n_sdim+1);
else
  xLDof = zeros(n_sdim,1);
end
sfun  = 'sf_disc0';


switch i_eval   % Evaluation type flag.

  case 1        % Evaluation of function values.
    vBase = 1;

  otherwise
    vBase = 0;
end
```

11.42 ellib/sf_disc1

SF_DISC1 Linear discontinuous shape function (P1).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_DISC1(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates discontinuous linear shape functions with with values defined by a constant and derivatives in the cell interior.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_disc0

Code listing

```
% Check for evaluation on simplicies.
 if( n_vert==n_sdim+1 )
   f_simplex = 1;
 else
   f_simplex = 0;
 end


 nLDof = [0 0 0 n_sdim+1];
 if( f_simplex )
```

```
    xLDof = ones(n_sdim+1,1)/(n_sdim+1);
  else
    xLDof = zeros(n_sdim,1);
  end
  xLDof = repmat(xLDof,1,n_sdim+1);
  sfun  = 'sf_disc1';


  switch i_eval   % Evaluation type flag.

    case {1}   % Evaluation of function values.

      if ( f_simplex || i_dof<=n_sdim )

        vBase = xi(i_dof);

      else

        vBase = 1;
      end

    case {2,3,4}   % Evaluation of first order derivatives.

      if ( f_simplex || i_dof<=n_sdim )

        i_col = i_dof+(i_eval-2)*(n_sdim+f_simplex);

        vBase = aInvJac(:,i_col);

      else

        vBase = 0;
      end

    otherwise
      vBase = 0;

  end
```

11.43 ellib/sf_hex_Q1

SF_HEX_Q1 Trilinear conforming shape function for hexahedrons (Q1).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_HEX_Q1(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming trilinear Q1 shape functions on hexahedrons with values defined in the nodes. XI is [-1..1]^3 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  3             Number of space dimensions
n_vert      scalar:  8             Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag1, sf_hex_Q1nc

Code listing

```
nLDof = [8 0 0 0];
xLDof = [-1  1  1 -1 -1  1  1 -1; ...
          -1 -1  1  1 -1 -1  1  1; ...
          -1 -1 -1 -1  1  1  1  1];
sfun  = 'sf_hex_Q1';


switch i_eval   % Evaluation type flag.

  case 1   % Evaluation of function values.

    switch i_dof   % Basis function to evaluate.

      case 1
        vBase = (1-xi(1))*(1-xi(2))*(1-xi(3))/8;
      case 2
        vBase = (1+xi(1))*(1-xi(2))*(1-xi(3))/8;
      case 3
        vBase = (1+xi(1))*(1+xi(2))*(1-xi(3))/8;
```

```
        case 4
          vBase = (1-xi(1))*(1+xi(2))*(1-xi(3))/8;
        case 5
          vBase = (1-xi(1))*(1-xi(2))*(1+xi(3))/8;
        case 6
          vBase = (1+xi(1))*(1-xi(2))*(1+xi(3))/8;
        case 7
          vBase = (1+xi(1))*(1+xi(2))*(1+xi(3))/8;
        case 8
          vBase = (1-xi(1))*(1+xi(2))*(1+xi(3))/8;
      end

    case {2,3,4}   % Evaluation of first order derivatives.

      switch i_dof   % Basis function to evaluate.

        case 1
          dNdxi1 = -(1-xi(2))*(1-xi(3))/8;
          dNdxi2 = -(1-xi(1))*(1-xi(3))/8;
          dNdxi3 = -(1-xi(1))*(1-xi(2))/8;
        case 2
          dNdxi1 =  (1-xi(2))*(1-xi(3))/8;
          dNdxi2 = -(1+xi(1))*(1-xi(3))/8;
          dNdxi3 = -(1+xi(1))*(1-xi(2))/8;
        case 3
          dNdxi1 =  (1+xi(2))*(1-xi(3))/8;
          dNdxi2 =  (1+xi(1))*(1-xi(3))/8;
          dNdxi3 = -(1+xi(1))*(1+xi(2))/8;
        case 4
          dNdxi1 = -(1+xi(2))*(1-xi(3))/8;
          dNdxi2 =  (1-xi(1))*(1-xi(3))/8;
          dNdxi3 = -(1-xi(1))*(1+xi(2))/8;
        case 5
          dNdxi1 = -(1-xi(2))*(1+xi(3))/8;
          dNdxi2 = -(1-xi(1))*(1+xi(3))/8;
          dNdxi3 =  (1-xi(1))*(1-xi(2))/8;
        case 6
          dNdxi1 =  (1-xi(2))*(1+xi(3))/8;
          dNdxi2 = -(1+xi(1))*(1+xi(3))/8;
          dNdxi3 =  (1+xi(1))*(1-xi(2))/8;
        case 7
          dNdxi1 =  (1+xi(2))*(1+xi(3))/8;
          dNdxi2 =  (1+xi(1))*(1+xi(3))/8;
          dNdxi3 =  (1+xi(1))*(1+xi(2))/8;
        case 8
          dNdxi1 = -(1+xi(2))*(1+xi(3))/8;
          dNdxi2 =  (1-xi(1))*(1+xi(3))/8;
          dNdxi3 =  (1-xi(1))*(1+xi(2))/8;
      end

      if     ( i_eval==2 )   % x-derivative.
```

```
      vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3;

    elseif ( i_eval==3 )   % y-derivative.

      vBase = aInvJac(:,4)*dNdxi1 + aInvJac(:,5)*dNdxi2 + aInvJac(:,6)*dNdxi3;

    elseif ( i_eval==4 )   % z-derivative.

      vBase = aInvJac(:,7)*dNdxi1 + aInvJac(:,8)*dNdxi2 + aInvJac(:,9)*dNdxi3;
    end

  case {22,23,24,32,33,34,42,43,44}   % Evaluation of second order derivatives.

    if( any(any(abs([aInvJac(:,[2 3 4 6 7 8])])>eps*1e2)) )
      warning('sf_hex_Q1: 2nd derivatives for non-rectangular cells shapes not supported.')
    end

    switch i_dof

      case 1
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 1/8 - xi(3)/8;
        d2Ndxi3dxi1 = 1/8 - xi(2)/8;
        d2Ndxi1dxi2 = 1/8 - xi(3)/8;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 1/8 - xi(1)/8;
        d2Ndxi1dxi3 = 1/8 - xi(2)/8;
        d2Ndxi2dxi3 = 1/8 - xi(1)/8;
        d2Ndxi3dxi3 = 0;

      case 2
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = xi(3)/8 - 1/8;
        d2Ndxi3dxi1 = xi(2)/8 - 1/8;
        d2Ndxi1dxi2 = xi(3)/8 - 1/8;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = xi(1)/8 + 1/8;
        d2Ndxi1dxi3 = xi(2)/8 - 1/8;
        d2Ndxi2dxi3 = xi(1)/8 + 1/8;
        d2Ndxi3dxi3 = 0;

      case 3
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 1/8 - xi(3)/8;
        d2Ndxi3dxi1 = - xi(2)/8 - 1/8;
        d2Ndxi1dxi2 = 1/8 - xi(3)/8;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = - xi(1)/8 - 1/8;
        d2Ndxi1dxi3 = - xi(2)/8 - 1/8;
        d2Ndxi2dxi3 = - xi(1)/8 - 1/8;
        d2Ndxi3dxi3 = 0;
```

```
        case 4
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = xi(3)/8 - 1/8;
          d2Ndxi3dxi1 = xi(2)/8 + 1/8;
          d2Ndxi1dxi2 = xi(3)/8 - 1/8;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = xi(1)/8 - 1/8;
          d2Ndxi1dxi3 = xi(2)/8 + 1/8;
          d2Ndxi2dxi3 = xi(1)/8 - 1/8;
          d2Ndxi3dxi3 = 0;

        case 5
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = xi(3)/8 + 1/8;
          d2Ndxi3dxi1 = xi(2)/8 - 1/8;
          d2Ndxi1dxi2 = xi(3)/8 + 1/8;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = xi(1)/8 - 1/8;
          d2Ndxi1dxi3 = xi(2)/8 - 1/8;
          d2Ndxi2dxi3 = xi(1)/8 - 1/8;
          d2Ndxi3dxi3 = 0;

        case 6
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = - xi(3)/8 - 1/8;
          d2Ndxi3dxi1 = 1/8 - xi(2)/8;
          d2Ndxi1dxi2 = - xi(3)/8 - 1/8;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = - xi(1)/8 - 1/8;
          d2Ndxi1dxi3 = 1/8 - xi(2)/8;
          d2Ndxi2dxi3 = - xi(1)/8 - 1/8;
          d2Ndxi3dxi3 = 0;

        case 7
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = xi(3)/8 + 1/8;
          d2Ndxi3dxi1 = xi(2)/8 + 1/8;
          d2Ndxi1dxi2 = xi(3)/8 + 1/8;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = xi(1)/8 + 1/8;
          d2Ndxi1dxi3 = xi(2)/8 + 1/8;
          d2Ndxi2dxi3 = xi(1)/8 + 1/8;
          d2Ndxi3dxi3 = 0;

        case 8
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = - xi(3)/8 - 1/8;
          d2Ndxi3dxi1 = - xi(2)/8 - 1/8;
          d2Ndxi1dxi2 = - xi(3)/8 - 1/8;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 1/8 - xi(1)/8;
          d2Ndxi1dxi3 = - xi(2)/8 - 1/8;
```

```
          d2Ndxi2dxi3 = 1/8 - xi(1)/8;
          d2Ndxi3dxi3 = 0;

      end

      switch( i_eval )
        case 22
          vBase = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
                  aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3
        case 33
          vBase = aInvJac(:,4).*( aInvJac(:,4)*d2Ndxi1dxi1 + aInvJac(:,5)*d2Ndxi2dxi1 + aInvJac(:,6
                  aInvJac(:,5).*( aInvJac(:,4)*d2Ndxi1dxi2 + aInvJac(:,5)*d2Ndxi2dxi2 + aInvJac(:,6
                  aInvJac(:,6).*( aInvJac(:,4)*d2Ndxi1dxi3 + aInvJac(:,5)*d2Ndxi2dxi3 + aInvJac(:,6

        case 44
          vBase = aInvJac(:,7).*( aInvJac(:,7)*d2Ndxi1dxi1 + aInvJac(:,8)*d2Ndxi2dxi1 + aInvJac(:,9
                  aInvJac(:,8).*( aInvJac(:,7)*d2Ndxi1dxi2 + aInvJac(:,8)*d2Ndxi2dxi2 + aInvJac(:,9
                  aInvJac(:,9).*( aInvJac(:,7)*d2Ndxi1dxi3 + aInvJac(:,8)*d2Ndxi2dxi3 + aInvJac(:,9

        case {23,32}
          vBase = aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,5).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
                  aInvJac(:,6).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3

        case {24,42}
          vBase = aInvJac(:,7).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,8).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
                  aInvJac(:,9).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3

        case {34,43}
          vBase = aInvJac(:,7).*( aInvJac(:,4)*d2Ndxi1dxi1 + aInvJac(:,5)*d2Ndxi2dxi1 + aInvJac(:,6
                  aInvJac(:,8).*( aInvJac(:,4)*d2Ndxi1dxi2 + aInvJac(:,5)*d2Ndxi2dxi2 + aInvJac(:,6
                  aInvJac(:,9).*( aInvJac(:,4)*d2Ndxi1dxi3 + aInvJac(:,5)*d2Ndxi2dxi3 + aInvJac(:,6
      end

     otherwise
       vBase = 0;

   end
```

11.44 ellib/sf_hex_Q1nc

SF_HEX_Q1NC Trilinear nonconforming shape function for hexahedrons (Q1∼).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_HEX_Q1NC(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates nonconforming trilinear Q1∼ shape functions on hexahedrons with values defined in the face centers.

XI is
[-1..1]^3 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  3             Number of space dimensions
n_vert      scalar:  8             Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                             .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag1, sf_hex_Q1

Code listing

```
nLDof = [0 0 6 0];
xLDof = [ 0  0  1 0 -1 0; ...
          0 -1  0 1  0 0; ...
         -1  0  0 0  0 1];
sfun  = 'sf_hex_Q1nc';


switch i_eval   % Evaluation type flag.

  case 1   % Evaluation of function values.

    switch i_dof   % Basis function to evaluate.

      case 1
        vBase = (xi(1)^2-xi(2)^2)/6-(xi(1)^2-xi(3)^2)/3-xi(3)/2+1/6;
      case 2
```

```
      vBase =-(xi(1)^2-xi(2)^2)/3+(xi(1)^2-xi(3)^2)/6-xi(2)/2+1/6;
    case 3
      vBase = (xi(1)^2-xi(2)^2)/6+(xi(1)^2-xi(3)^2)/6+xi(1)/2+1/6;
    case 4
      vBase =-(xi(1)^2-xi(2)^2)/3+(xi(1)^2-xi(3)^2)/6+xi(2)/2+1/6;
    case 5
      vBase = (xi(1)^2-xi(2)^2)/6+(xi(1)^2-xi(3)^2)/6-xi(1)/2+1/6;
    case 6
      vBase = (xi(1)^2-xi(2)^2)/6-(xi(1)^2-xi(3)^2)/3+xi(3)/2+1/6;
  end

case {2,3,4}   % Evaluation of first order derivatives.

  switch i_dof   % Basis function to evaluate.

    case 1
      dNdxi1 = -xi(1)/3;
      dNdxi2 = -xi(2)/3;
      dNdxi3 = 2/3*xi(3)-1/2;
    case 2
      dNdxi1 = -xi(1)/3;
      dNdxi2 = 2/3*xi(2)-1/2;
      dNdxi3 = -xi(3)/3;
    case 3
      dNdxi1 = 2/3*xi(1)+1/2;
      dNdxi2 = -xi(2)/3;
      dNdxi3 = -xi(3)/3;
    case 4
      dNdxi1 = -xi(1)/3;
      dNdxi2 = 2/3*xi(2)+1/2;
      dNdxi3 = -xi(3)/3;
    case 5
      dNdxi1 = 2/3*xi(1)-1/2;
      dNdxi2 = -xi(2)/3;
      dNdxi3 = -xi(3)/3;
    case 6
      dNdxi1 = -xi(1)/3;
      dNdxi2 = -xi(2)/3;
      dNdxi3 = 2/3*xi(3)+1/2;
  end

  if     ( i_eval==2 )   % x-derivative.

    vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3;

  elseif ( i_eval==3 )   % y-derivative.

    vBase = aInvJac(:,4)*dNdxi1 + aInvJac(:,5)*dNdxi2 + aInvJac(:,6)*dNdxi3;

  elseif ( i_eval==4 )   % z-derivative.

    vBase = aInvJac(:,7)*dNdxi1 + aInvJac(:,8)*dNdxi2 + aInvJac(:,9)*dNdxi3;
```

```
    end

  case {22,23,24,32,33,34,42,43,44}   % Evaluation of second order derivatives.

    if( any(any(abs([aInvJac(:,[2 3 4 6 7 8])])>eps*1e2)) )
      warning('sf_hex_Q1nc: 2nd derivatives for non-rectangular cells shapes not supported.')
    end

    switch i_dof

      case 1
        d2Ndxi1dxi1 = -1/3;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = -1/3;
        d2Ndxi3dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 2/3;

      case 2
        d2Ndxi1dxi1 = -1/3;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 2/3;
        d2Ndxi3dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = -1/3;

      case 3
        d2Ndxi1dxi1 = 2/3;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = -1/3;
        d2Ndxi3dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = -1/3;

      case 4
        d2Ndxi1dxi1 = -1/3;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 2/3;
        d2Ndxi3dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
```

```
          d2Ndxi3dxi3 = -1/3;

        case 5
          d2Ndxi1dxi1 = 2/3;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = -1/3;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = -1/3;

        case 6
          d2Ndxi1dxi1 = -1/3;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = -1/3;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 2/3;

      end

      switch( i_eval )
        case 22
          vBase = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
                  aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3
        case 33
          vBase = aInvJac(:,4).*( aInvJac(:,4)*d2Ndxi1dxi1 + aInvJac(:,5)*d2Ndxi2dxi1 + aInvJac(:,6
                  aInvJac(:,5).*( aInvJac(:,4)*d2Ndxi1dxi2 + aInvJac(:,5)*d2Ndxi2dxi2 + aInvJac(:,6
                  aInvJac(:,6).*( aInvJac(:,4)*d2Ndxi1dxi3 + aInvJac(:,5)*d2Ndxi2dxi3 + aInvJac(:,6

        case 44
          vBase = aInvJac(:,7).*( aInvJac(:,7)*d2Ndxi1dxi1 + aInvJac(:,8)*d2Ndxi2dxi1 + aInvJac(:,9
                  aInvJac(:,8).*( aInvJac(:,7)*d2Ndxi1dxi2 + aInvJac(:,8)*d2Ndxi2dxi2 + aInvJac(:,9
                  aInvJac(:,9).*( aInvJac(:,7)*d2Ndxi1dxi3 + aInvJac(:,8)*d2Ndxi2dxi3 + aInvJac(:,9

        case {23,32}
          vBase = aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,5).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
                  aInvJac(:,6).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3

        case {24,42}
          vBase = aInvJac(:,7).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,8).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
                  aInvJac(:,9).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3

        case {34,43}
```

```
          vBase = aInvJac(:,7).*( aInvJac(:,4)*d2Ndxi1dxi1 + aInvJac(:,5)*d2Ndxi2dxi1 + aInvJac(:,6
                  aInvJac(:,8).*( aInvJac(:,4)*d2Ndxi1dxi2 + aInvJac(:,5)*d2Ndxi2dxi2 + aInvJac(:,6
                  aInvJac(:,9).*( aInvJac(:,4)*d2Ndxi1dxi3 + aInvJac(:,5)*d2Ndxi2dxi3 + aInvJac(:,6
      end

    otherwise
      vBase = 0;

  end
```

11.45 ellib/sf_hex_Q2

SF_HEX_Q2 Triquadratic conforming shape function for hexahedrons (Q2).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_HEX_Q2(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming triquadratic Q2 shape functions on hexahedrons with values defined in the nodes, edges, faces, and cell center. XI is
[-1..1]^3 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  3             Number of space dimensions
n_vert      scalar:  8             Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag2, sf_hex_Q1

Code listing

```
 nLDof = [8 12 6 1];
 xLDof = [-1  1  1 -1  -1  1  1 -1   0  1  0 -1  -1  1  1 -1   0  1  0 -1   0  0  1 0 -1 0
0; ...
          -1 -1  1  1  -1 -1  1  1  -1  0  1  0  -1 -1  1  1  -1  0  1  0   0 -1  0 1
0 0  0; ...
          -1 -1 -1 -1   1  1  1  1  -1 -1 -1 -1   0  0  0  0   1  1  1  1  -1  0  0 0
0 1  0];
 sfun  = 'sf_hex_Q2';


 switch i_eval   % Evaluation type flag.

   case 1   % Evaluation of function values.

     switch i_dof   % Basis function to evaluate.

       case 1
         vBase = -xi(1)*(1-xi(1))*xi(2)*(1-xi(2))*xi(3)*(1-xi(3))/8;
       case 2
         vBase =  xi(1)*(1+xi(1))*xi(2)*(1-xi(2))*xi(3)*(1-xi(3))/8;
       case 3
         vBase = -xi(1)*(1+xi(1))*xi(2)*(1+xi(2))*xi(3)*(1-xi(3))/8;
       case 4
         vBase =  xi(1)*(1-xi(1))*xi(2)*(1+xi(2))*xi(3)*(1-xi(3))/8;
       case 5
         vBase =  xi(1)*(1-xi(1))*xi(2)*(1-xi(2))*xi(3)*(1+xi(3))/8;
       case 6
         vBase = -xi(1)*(1+xi(1))*xi(2)*(1-xi(2))*xi(3)*(1+xi(3))/8;
       case 7
         vBase =  xi(1)*(1+xi(1))*xi(2)*(1+xi(2))*xi(3)*(1+xi(3))/8;
       case 8
         vBase = -xi(1)*(1-xi(1))*xi(2)*(1+xi(2))*xi(3)*(1+xi(3))/8;
       case 9
         vBase =  (2-2*xi(1)^2)*xi(2)*(1-xi(2))*xi(3)*(1-xi(3))/8;
       case 10
         vBase = -xi(1)*(1+xi(1))*(2-2*xi(2)^2)*xi(3)*(1-xi(3))/8;
       case 11
         vBase = -(2-2*xi(1)^2)*xi(2)*(1+xi(2))*xi(3)*(1-xi(3))/8;
       case 12
         vBase =  xi(1)*(1-xi(1))*(2-2*xi(2)^2)*xi(3)*(1-xi(3))/8;
       case 13
         vBase =  xi(1)*(1-xi(1))*xi(2)*(1-xi(2))*(2-2*xi(3)^2)/8;
```

```
        case 14
          vBase = -xi(1)*(1+xi(1))*xi(2)*(1-xi(2))*(2-2*xi(3)^2)/8;
        case 15
          vBase =  xi(1)*(1+xi(1))*xi(2)*(1+xi(2))*(2-2*xi(3)^2)/8;
        case 16
          vBase = -xi(1)*(1-xi(1))*xi(2)*(1+xi(2))*(2-2*xi(3)^2)/8;
        case 17
          vBase = -(2-2*xi(1)^2)*xi(2)*(1-xi(2))*xi(3)*(1+xi(3))/8;
        case 18
          vBase =  xi(1)*(1+xi(1))*(2-2*xi(2)^2)*xi(3)*(1+xi(3))/8;
        case 19
          vBase =  (2-2*xi(1)^2)*xi(2)*(1+xi(2))*xi(3)*(1+xi(3))/8;
        case 20
          vBase = -xi(1)*(1-xi(1))*(2-2*xi(2)^2)*xi(3)*(1+xi(3))/8;
        case 21
          vBase = -(2-2*xi(1)^2)*(2-2*xi(2)^2)*xi(3)*(1-xi(3))/8;
        case 22
          vBase = -(2-2*xi(1)^2)*xi(2)*(1-xi(2))*(2-2*xi(3)^2)/8;
        case 23
          vBase =  xi(1)*(1+xi(1))*(2-2*xi(2)^2)*(2-2*xi(3)^2)/8;
        case 24
          vBase =  (2-2*xi(1)^2)*xi(2)*(1+xi(2))*(2-2*xi(3)^2)/8;
        case 25
          vBase = -xi(1)*(1-xi(1))*(2-2*xi(2)^2)*(2-2*xi(3)^2)/8;
        case 26
          vBase =  (2-2*xi(1)^2)*(2-2*xi(2)^2)*xi(3)*(1+xi(3))/8;
        case 27
          vBase =  (2-2*xi(1)^2)*(2-2*xi(2)^2)*(2-2*xi(3)^2)/8;

      end

    case {2,3,4}   % Evaluation of first order derivatives.

      switch i_dof   % Basis function to evaluate.
        case 1
          dNdxi1 = (xi(2)*xi(3)*(2*xi(1)-1)*(xi(2)-1)*(xi(3)-1))/8;
          dNdxi2 = (xi(1)*xi(3)*(2*xi(2)-1)*(xi(1)-1)*(xi(3)-1))/8;
          dNdxi3 = (xi(1)*xi(2)*(2*xi(3)-1)*(xi(1)-1)*(xi(2)-1))/8;
        case 2
          dNdxi1 = (xi(2)*xi(3)*(2*xi(1)+1)*(xi(2)-1)*(xi(3)-1))/8;
          dNdxi2 = (xi(1)*xi(3)*(2*xi(2)-1)*(xi(1)+1)*(xi(3)-1))/8;
          dNdxi3 = (xi(1)*xi(2)*(2*xi(3)-1)*(xi(1)+1)*(xi(2)-1))/8;
        case 3
          dNdxi1 = (xi(2)*xi(3)*(2*xi(1)+1)*(xi(2)+1)*(xi(3)-1))/8;
          dNdxi2 = (xi(1)*xi(3)*(2*xi(2)+1)*(xi(1)+1)*(xi(3)-1))/8;
          dNdxi3 = (xi(1)*xi(2)*(2*xi(3)-1)*(xi(1)+1)*(xi(2)+1))/8;
        case 4
          dNdxi1 = (xi(2)*xi(3)*(2*xi(1)-1)*(xi(2)+1)*(xi(3)-1))/8;
          dNdxi2 = (xi(1)*xi(3)*(2*xi(2)+1)*(xi(1)-1)*(xi(3)-1))/8;
          dNdxi3 = (xi(1)*xi(2)*(2*xi(3)-1)*(xi(1)-1)*(xi(2)+1))/8;
        case 5
          dNdxi1 = (xi(2)*xi(3)*(2*xi(1)-1)*(xi(2)-1)*(xi(3)+1))/8;
```

```
    dNdxi2 = (xi(1)*xi(3)*(2*xi(2)-1)*(xi(1)-1)*(xi(3)+1))/8;
    dNdxi3 = (xi(1)*xi(2)*(2*xi(3)+1)*(xi(1)-1)*(xi(2)-1))/8;
  case 6
    dNdxi1 = (xi(2)*xi(3)*(2*xi(1)+1)*(xi(2)-1)*(xi(3)+1))/8;
    dNdxi2 = (xi(1)*xi(3)*(2*xi(2)-1)*(xi(1)+1)*(xi(3)+1))/8;
    dNdxi3 = (xi(1)*xi(2)*(2*xi(3)+1)*(xi(1)+1)*(xi(2)-1))/8;
  case 7
    dNdxi1 = (xi(2)*xi(3)*(2*xi(1)+1)*(xi(2)+1)*(xi(3)+1))/8;
    dNdxi2 = (xi(1)*xi(3)*(2*xi(2)+1)*(xi(1)+1)*(xi(3)+1))/8;
    dNdxi3 = (xi(1)*xi(2)*(2*xi(3)+1)*(xi(1)+1)*(xi(2)+1))/8;
  case 8
    dNdxi1 = (xi(2)*xi(3)*(2*xi(1)-1)*(xi(2)+1)*(xi(3)+1))/8;
    dNdxi2 = (xi(1)*xi(3)*(2*xi(2)+1)*(xi(1)-1)*(xi(3)+1))/8;
    dNdxi3 = (xi(1)*xi(2)*(2*xi(3)+1)*(xi(1)-1)*(xi(2)+1))/8;
  case 9
    dNdxi1 = -(xi(1)*xi(2)*xi(3)*(xi(2)-1)*(xi(3)-1))/2;
    dNdxi2 = -(xi(3)*(2*xi(2)-1)*(xi(1)^2-1)*(xi(3)-1))/4;
    dNdxi3 = -(xi(2)*(2*xi(3)-1)*(xi(1)^2-1)*(xi(2)-1))/4;
  case 10
    dNdxi1 = -(xi(3)*(2*xi(1)+1)*(xi(2)^2-1)*(xi(3)-1))/4;
    dNdxi2 = -(xi(1)*xi(2)*xi(3)*(xi(1)+1)*(xi(3)-1))/2;
    dNdxi3 = -(xi(1)*(2*xi(3)-1)*(xi(2)^2-1)*(xi(1)+1))/4;
  case 11
    dNdxi1 = -(xi(1)*xi(2)*xi(3)*(xi(2)+1)*(xi(3)-1))/2;
    dNdxi2 = -(xi(3)*(2*xi(2)+1)*(xi(1)^2-1)*(xi(3)-1))/4;
    dNdxi3 = -(xi(2)*(2*xi(3)-1)*(xi(1)^2-1)*(xi(2)+1))/4;
  case 12
    dNdxi1 = -(xi(3)*(2*xi(1)-1)*(xi(2)^2-1)*(xi(3)-1))/4;
    dNdxi2 = -(xi(1)*xi(2)*xi(3)*(xi(1)-1)*(xi(3)-1))/2;
    dNdxi3 = -(xi(1)*(2*xi(3)-1)*(xi(2)^2-1)*(xi(1)-1))/4;
  case 13
    dNdxi1 = -(xi(2)*(2*xi(1)-1)*(xi(3)^2-1)*(xi(2)-1))/4;
    dNdxi2 = -(xi(1)*(2*xi(2)-1)*(xi(3)^2-1)*(xi(1)-1))/4;
    dNdxi3 = -(xi(1)*xi(2)*xi(3)*(xi(1)-1)*(xi(2)-1))/2;
  case 14
    dNdxi1 = -(xi(2)*(2*xi(1)+1)*(xi(3)^2-1)*(xi(2)-1))/4;
    dNdxi2 = -(xi(1)*(2*xi(2)-1)*(xi(3)^2-1)*(xi(1)+1))/4;
    dNdxi3 = -(xi(1)*xi(2)*xi(3)*(xi(1)+1)*(xi(2)-1))/2;
  case 15
    dNdxi1 = -(xi(2)*(2*xi(1)+1)*(xi(3)^2-1)*(xi(2)+1))/4;
    dNdxi2 = -(xi(1)*(2*xi(2)+1)*(xi(3)^2-1)*(xi(1)+1))/4;
    dNdxi3 = -(xi(1)*xi(2)*xi(3)*(xi(1)+1)*(xi(2)+1))/2;
  case 16
    dNdxi1 = -(xi(2)*(2*xi(1)-1)*(xi(3)^2-1)*(xi(2)+1))/4;
    dNdxi2 = -(xi(1)*(2*xi(2)+1)*(xi(3)^2-1)*(xi(1)-1))/4;
    dNdxi3 = -(xi(1)*xi(2)*xi(3)*(xi(1)-1)*(xi(2)+1))/2;
  case 17
    dNdxi1 = -(xi(1)*xi(2)*xi(3)*(xi(2)-1)*(xi(3)+1))/2;
    dNdxi2 = -(xi(3)*(2*xi(2)-1)*(xi(1)^2-1)*(xi(3)+1))/4;
    dNdxi3 = -(xi(2)*(2*xi(3)+1)*(xi(1)^2-1)*(xi(2)-1))/4;
  case 18
    dNdxi1 = -(xi(3)*(2*xi(1)+1)*(xi(2)^2-1)*(xi(3)+1))/4;
```

```
      dNdxi2 = -(xi(1)*xi(2)*xi(3)*(xi(1)+1)*(xi(3)+1))/2;
      dNdxi3 = -(xi(1)*(2*xi(3)+1)*(xi(2)^2-1)*(xi(1)+1))/4;
    case 19
      dNdxi1 = -(xi(1)*xi(2)*xi(3)*(xi(2)+1)*(xi(3)+1))/2;
      dNdxi2 = -(xi(3)*(2*xi(2)+1)*(xi(1)^2-1)*(xi(3)+1))/4;
      dNdxi3 = -(xi(2)*(2*xi(3)+1)*(xi(1)^2-1)*(xi(2)+1))/4;
    case 20
      dNdxi1 = -(xi(3)*(2*xi(1)-1)*(xi(2)^2-1)*(xi(3)+1))/4;
      dNdxi2 = -(xi(1)*xi(2)*xi(3)*(xi(1)-1)*(xi(3)+1))/2;
      dNdxi3 = -(xi(1)*(2*xi(3)+1)*(xi(2)^2-1)*(xi(1)-1))/4;
    case 21
      dNdxi1 = xi(1)*xi(3)*(xi(2)^2-1)*(xi(3)-1);
      dNdxi2 = xi(2)*xi(3)*(xi(1)^2-1)*(xi(3)-1);
      dNdxi3 = ((2*xi(3)-1)*(xi(1)^2-1)*(xi(2)^2-1))/2;
    case 22
      dNdxi1 = xi(1)*xi(2)*(xi(3)^2-1)*(xi(2)-1);
      dNdxi2 = ((2*xi(2)-1)*(xi(1)^2-1)*(xi(3)^2-1))/2;
      dNdxi3 = xi(2)*xi(3)*(xi(1)^2-1)*(xi(2)-1);
    case 23
      dNdxi1 = ((2*xi(1)+1)*(xi(2)^2-1)*(xi(3)^2-1))/2;
      dNdxi2 = xi(1)*xi(2)*(xi(3)^2-1)*(xi(1)+1);
      dNdxi3 = xi(1)*xi(3)*(xi(2)^2-1)*(xi(1)+1);
    case 24
      dNdxi1 = xi(1)*xi(2)*(xi(3)^2-1)*(xi(2)+1);
      dNdxi2 = ((2*xi(2)+1)*(xi(1)^2-1)*(xi(3)^2-1))/2;
      dNdxi3 = xi(2)*xi(3)*(xi(1)^2-1)*(xi(2)+1);
    case 25
      dNdxi1 = ((2*xi(1)-1)*(xi(2)^2-1)*(xi(3)^2-1))/2;
      dNdxi2 = xi(1)*xi(2)*(xi(3)^2-1)*(xi(1)-1);
      dNdxi3 = xi(1)*xi(3)*(xi(2)^2-1)*(xi(1)-1);
    case 26
      dNdxi1 = xi(1)*xi(3)*(xi(2)^2-1)*(xi(3)+1);
      dNdxi2 = xi(2)*xi(3)*(xi(1)^2-1)*(xi(3)+1);
      dNdxi3 = ((2*xi(3)+1)*(xi(1)^2-1)*(xi(2)^2-1))/2;
    case 27
      dNdxi1 = -2*xi(1)*(xi(2)^2-1)*(xi(3)^2-1);
      dNdxi2 = -2*xi(2)*(xi(1)^2-1)*(xi(3)^2-1);
      dNdxi3 = -2*xi(3)*(xi(1)^2-1)*(xi(2)^2-1);
  end

  if     ( i_eval==2 )   % x-derivative.

    vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3;

  elseif ( i_eval==3 )   % y-derivative.

    vBase = aInvJac(:,4)*dNdxi1 + aInvJac(:,5)*dNdxi2 + aInvJac(:,6)*dNdxi3;

  elseif ( i_eval==4 )   % z-derivative.

    vBase = aInvJac(:,7)*dNdxi1 + aInvJac(:,8)*dNdxi2 + aInvJac(:,9)*dNdxi3;
  end
```

```
case {22,23,24,32,33,34,42,43,44}   % Evaluation of second order derivatives.

  if( any(any(abs([aInvJac(:,[2 3 4 6 7 8])])>eps*1e2)) )
    warning('sf_hex_Q2: 2nd derivatives for non-rectangular cells shapes not supported.')
  end

  switch i_dof

    case 1
      d2Ndxi1dxi1 = (xi(2)*xi(3)*(xi(2) - 1)*(xi(3) - 1))/4;
      d2Ndxi2dxi1 = (xi(3)*(2*xi(1) - 1)*(2*xi(2) - 1)*(xi(3) - 1))/8;
      d2Ndxi3dxi1 = (xi(2)*(2*xi(1) - 1)*(2*xi(3) - 1)*(xi(2) - 1))/8;
      d2Ndxi1dxi2 = (xi(3)*(2*xi(1) - 1)*(2*xi(2) - 1)*(xi(3) - 1))/8;
      d2Ndxi2dxi2 = (xi(1)*xi(3)*(xi(1) - 1)*(xi(3) - 1))/4;
      d2Ndxi3dxi2 = (xi(1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) - 1))/8;
      d2Ndxi1dxi3 = (xi(2)*(2*xi(1) - 1)*(2*xi(3) - 1)*(xi(2) - 1))/8;
      d2Ndxi2dxi3 = (xi(1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) - 1))/8;
      d2Ndxi3dxi3 = (xi(1)*xi(2)*(xi(1) - 1)*(xi(2) - 1))/4;

    case 2
      d2Ndxi1dxi1 = (xi(2)*xi(3)*(xi(2) - 1)*(xi(3) - 1))/4;
      d2Ndxi2dxi1 = (xi(3)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(3) - 1))/8;
      d2Ndxi3dxi1 = (xi(2)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(2) - 1))/8;
      d2Ndxi1dxi2 = (xi(3)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(3) - 1))/8;
      d2Ndxi2dxi2 = (xi(1)*xi(3)*(xi(1) + 1)*(xi(3) - 1))/4;
      d2Ndxi3dxi2 = (xi(1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) + 1))/8;
      d2Ndxi1dxi3 = (xi(2)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(2) - 1))/8;
      d2Ndxi2dxi3 = (xi(1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) + 1))/8;
      d2Ndxi3dxi3 = (xi(1)*xi(2)*(xi(1) + 1)*(xi(2) - 1))/4;

    case 3
      d2Ndxi1dxi1 = (xi(2)*xi(3)*(xi(2) + 1)*(xi(3) - 1))/4;
      d2Ndxi2dxi1 = (xi(3)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(3) - 1))/8;
      d2Ndxi3dxi1 = (xi(2)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(2) + 1))/8;
      d2Ndxi1dxi2 = (xi(3)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(3) - 1))/8;
      d2Ndxi2dxi2 = (xi(1)*xi(3)*(xi(1) + 1)*(xi(3) - 1))/4;
      d2Ndxi3dxi2 = (xi(1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) + 1))/8;
      d2Ndxi1dxi3 = (xi(2)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(2) + 1))/8;
      d2Ndxi2dxi3 = (xi(1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) + 1))/8;
      d2Ndxi3dxi3 = (xi(1)*xi(2)*(xi(1) + 1)*(xi(2) + 1))/4;

    case 4
      d2Ndxi1dxi1 = (xi(2)*xi(3)*(xi(2) + 1)*(xi(3) - 1))/4;
      d2Ndxi2dxi1 = (xi(3)*(2*xi(1) - 1)*(2*xi(2) + 1)*(xi(3) - 1))/8;
      d2Ndxi3dxi1 = (xi(2)*(2*xi(1) - 1)*(2*xi(3) - 1)*(xi(2) + 1))/8;
      d2Ndxi1dxi2 = (xi(3)*(2*xi(1) - 1)*(2*xi(2) + 1)*(xi(3) - 1))/8;
      d2Ndxi2dxi2 = (xi(1)*xi(3)*(xi(1) - 1)*(xi(3) - 1))/4;
      d2Ndxi3dxi2 = (xi(1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) - 1))/8;
      d2Ndxi1dxi3 = (xi(2)*(2*xi(1) - 1)*(2*xi(3) - 1)*(xi(2) + 1))/8;
      d2Ndxi2dxi3 = (xi(1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) - 1))/8;
      d2Ndxi3dxi3 = (xi(1)*xi(2)*(xi(1) - 1)*(xi(2) + 1))/4;
```

```
        case 5
          d2Ndxi1dxi1 = (xi(2)*xi(3)*(xi(2) - 1)*(xi(3) + 1))/4;
          d2Ndxi2dxi1 = (xi(3)*(2*xi(1) - 1)*(2*xi(2) - 1)*(xi(3) + 1))/8;
          d2Ndxi3dxi1 = (xi(2)*(2*xi(1) - 1)*(2*xi(3) + 1)*(xi(2) - 1))/8;
          d2Ndxi1dxi2 = (xi(3)*(2*xi(1) - 1)*(2*xi(2) - 1)*(xi(3) + 1))/8;
          d2Ndxi2dxi2 = (xi(1)*xi(3)*(xi(1) - 1)*(xi(3) + 1))/4;
          d2Ndxi3dxi2 = (xi(1)*(2*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) - 1))/8;
          d2Ndxi1dxi3 = (xi(2)*(2*xi(1) - 1)*(2*xi(3) + 1)*(xi(2) - 1))/8;
          d2Ndxi2dxi3 = (xi(1)*(2*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) - 1))/8;
          d2Ndxi3dxi3 = (xi(1)*xi(2)*(xi(1) - 1)*(xi(2) - 1))/4;

        case 6
          d2Ndxi1dxi1 = (xi(2)*xi(3)*(xi(2) - 1)*(xi(3) + 1))/4;
          d2Ndxi2dxi1 = (xi(3)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(3) + 1))/8;
          d2Ndxi3dxi1 = (xi(2)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(2) - 1))/8;
          d2Ndxi1dxi2 = (xi(3)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(3) + 1))/8;
          d2Ndxi2dxi2 = (xi(1)*xi(3)*(xi(1) + 1)*(xi(3) + 1))/4;
          d2Ndxi3dxi2 = (xi(1)*(2*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) + 1))/8;
          d2Ndxi1dxi3 = (xi(2)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(2) - 1))/8;
          d2Ndxi2dxi3 = (xi(1)*(2*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) + 1))/8;
          d2Ndxi3dxi3 = (xi(1)*xi(2)*(xi(1) + 1)*(xi(2) - 1))/4;

        case 7
          d2Ndxi1dxi1 = (xi(2)*xi(3)*(xi(2) + 1)*(xi(3) + 1))/4;
          d2Ndxi2dxi1 = (xi(3)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(3) + 1))/8;
          d2Ndxi3dxi1 = (xi(2)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(2) + 1))/8;
          d2Ndxi1dxi2 = (xi(3)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(3) + 1))/8;
          d2Ndxi2dxi2 = (xi(1)*xi(3)*(xi(1) + 1)*(xi(3) + 1))/4;
          d2Ndxi3dxi2 = (xi(1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) + 1))/8;
          d2Ndxi1dxi3 = (xi(2)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(2) + 1))/8;
          d2Ndxi2dxi3 = (xi(1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) + 1))/8;
          d2Ndxi3dxi3 = (xi(1)*xi(2)*(xi(1) + 1)*(xi(2) + 1))/4;

        case 8
          d2Ndxi1dxi1 = (xi(2)*xi(3)*(xi(2) + 1)*(xi(3) + 1))/4;
          d2Ndxi2dxi1 = (xi(3)*(2*xi(1) - 1)*(2*xi(2) + 1)*(xi(3) + 1))/8;
          d2Ndxi3dxi1 = (xi(2)*(2*xi(1) - 1)*(2*xi(3) + 1)*(xi(2) + 1))/8;
          d2Ndxi1dxi2 = (xi(3)*(2*xi(1) - 1)*(2*xi(2) + 1)*(xi(3) + 1))/8;
          d2Ndxi2dxi2 = (xi(1)*xi(3)*(xi(1) - 1)*(xi(3) + 1))/4;
          d2Ndxi3dxi2 = (xi(1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) - 1))/8;
          d2Ndxi1dxi3 = (xi(2)*(2*xi(1) - 1)*(2*xi(3) + 1)*(xi(2) + 1))/8;
          d2Ndxi2dxi3 = (xi(1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) - 1))/8;
          d2Ndxi3dxi3 = (xi(1)*xi(2)*(xi(1) - 1)*(xi(2) + 1))/4;

        case 9
          d2Ndxi1dxi1 = -(xi(2)*xi(3)*(xi(2) - 1)*(xi(3) - 1))/2;
          d2Ndxi2dxi1 = -(xi(1)*xi(3)*(2*xi(2) - 1)*(xi(3) - 1))/2;
          d2Ndxi3dxi1 = -(xi(1)*xi(2)*(2*xi(3) - 1)*(xi(2) - 1))/2;
          d2Ndxi1dxi2 = -(xi(1)*xi(3)*(2*xi(2) - 1)*(xi(3) - 1))/2;
          d2Ndxi2dxi2 = -(xi(3)*(xi(1)^2 - 1)*(xi(3) - 1))/2;
          d2Ndxi3dxi2 = -((2*xi(2) - 1)*(2*xi(3) - 1)*(xi(1)^2 - 1))/4;
```

```
    d2Ndxi1dxi3 = -(xi(1)*xi(2)*(2*xi(3) - 1)*(xi(2) - 1))/2;
    d2Ndxi2dxi3 = -((2*xi(2) - 1)*(2*xi(3) - 1)*(xi(1)^2 - 1))/4;
    d2Ndxi3dxi3 = -(xi(2)*(xi(1)^2 - 1)*(xi(2) - 1))/2;

  case 10
    d2Ndxi1dxi1 = -(xi(3)*(xi(2)^2 - 1)*(xi(3) - 1))/2;
    d2Ndxi2dxi1 = -(xi(2)*xi(3)*(2*xi(1) + 1)*(xi(3) - 1))/2;
    d2Ndxi3dxi1 = -((2*xi(1) + 1)*(2*xi(3) - 1)*(xi(2)^2 - 1))/4;
    d2Ndxi1dxi2 = -(xi(2)*xi(3)*(2*xi(1) + 1)*(xi(3) - 1))/2;
    d2Ndxi2dxi2 = -(xi(1)*xi(3)*(xi(1) + 1)*(xi(3) - 1))/2;
    d2Ndxi3dxi2 = -(xi(1)*xi(2)*(2*xi(3) - 1)*(xi(1) + 1))/2;
    d2Ndxi1dxi3 = -((2*xi(1) + 1)*(2*xi(3) - 1)*(xi(2)^2 - 1))/4;
    d2Ndxi2dxi3 = -(xi(1)*xi(2)*(2*xi(3) - 1)*(xi(1) + 1))/2;
    d2Ndxi3dxi3 = -(xi(1)*(xi(2)^2 - 1)*(xi(1) + 1))/2;

  case 11
    d2Ndxi1dxi1 = -(xi(2)*xi(3)*(xi(2) + 1)*(xi(3) - 1))/2;
    d2Ndxi2dxi1 = -(xi(1)*xi(3)*(2*xi(2) + 1)*(xi(3) - 1))/2;
    d2Ndxi3dxi1 = -(xi(1)*xi(2)*(2*xi(3) - 1)*(xi(2) + 1))/2;
    d2Ndxi1dxi2 = -(xi(1)*xi(3)*(2*xi(2) + 1)*(xi(3) - 1))/2;
    d2Ndxi2dxi2 = -(xi(3)*(xi(1)^2 - 1)*(xi(3) - 1))/2;
    d2Ndxi3dxi2 = -((2*xi(2) + 1)*(2*xi(3) - 1)*(xi(1)^2 - 1))/4;
    d2Ndxi1dxi3 = -(xi(1)*xi(2)*(2*xi(3) - 1)*(xi(2) + 1))/2;
    d2Ndxi2dxi3 = -((2*xi(2) + 1)*(2*xi(3) - 1)*(xi(1)^2 - 1))/4;
    d2Ndxi3dxi3 = -(xi(2)*(xi(1)^2 - 1)*(xi(2) + 1))/2;

  case 12
    d2Ndxi1dxi1 = -(xi(3)*(xi(2)^2 - 1)*(xi(3) - 1))/2;
    d2Ndxi2dxi1 = -(xi(2)*xi(3)*(2*xi(1) - 1)*(xi(3) - 1))/2;
    d2Ndxi3dxi1 = -((2*xi(1) - 1)*(2*xi(3) - 1)*(xi(2)^2 - 1))/4;
    d2Ndxi1dxi2 = -(xi(2)*xi(3)*(2*xi(1) - 1)*(xi(3) - 1))/2;
    d2Ndxi2dxi2 = -(xi(1)*xi(3)*(xi(1) - 1)*(xi(3) - 1))/2;
    d2Ndxi3dxi2 = -(xi(1)*xi(2)*(2*xi(3) - 1)*(xi(1) - 1))/2;
    d2Ndxi1dxi3 = -((2*xi(1) - 1)*(2*xi(3) - 1)*(xi(2)^2 - 1))/4;
    d2Ndxi2dxi3 = -(xi(1)*xi(2)*(2*xi(3) - 1)*(xi(1) - 1))/2;
    d2Ndxi3dxi3 = -(xi(1)*(xi(2)^2 - 1)*(xi(1) - 1))/2;

  case 13
    d2Ndxi1dxi1 = -(xi(2)*(xi(3)^2 - 1)*(xi(2) - 1))/2;
    d2Ndxi2dxi1 = -((2*xi(1) - 1)*(2*xi(2) - 1)*(xi(3)^2 - 1))/4;
    d2Ndxi3dxi1 = -(xi(2)*xi(3)*(2*xi(1) - 1)*(xi(2) - 1))/2;
    d2Ndxi1dxi2 = -((2*xi(1) - 1)*(2*xi(2) - 1)*(xi(3)^2 - 1))/4;
    d2Ndxi2dxi2 = -(xi(1)*(xi(3)^2 - 1)*(xi(1) - 1))/2;
    d2Ndxi3dxi2 = -(xi(1)*xi(3)*(2*xi(2) - 1)*(xi(1) - 1))/2;
    d2Ndxi1dxi3 = -(xi(2)*xi(3)*(2*xi(1) - 1)*(xi(2) - 1))/2;
    d2Ndxi2dxi3 = -(xi(1)*xi(3)*(2*xi(2) - 1)*(xi(1) - 1))/2;
    d2Ndxi3dxi3 = -(xi(1)*xi(2)*(xi(1) - 1)*(xi(2) - 1))/2;

  case 14
    d2Ndxi1dxi1 = -(xi(2)*(xi(3)^2 - 1)*(xi(2) - 1))/2;
    d2Ndxi2dxi1 = -((2*xi(1) + 1)*(2*xi(2) - 1)*(xi(3)^2 - 1))/4;
    d2Ndxi3dxi1 = -(xi(2)*xi(3)*(2*xi(1) + 1)*(xi(2) - 1))/2;
```

```
    d2Ndxi1dxi2 = -((2*xi(1) + 1)*(2*xi(2) - 1)*(xi(3)^2 - 1))/4;
    d2Ndxi2dxi2 = -(xi(1)*(xi(3)^2 - 1)*(xi(1) + 1))/2;
    d2Ndxi3dxi2 = -(xi(1)*xi(3)*(2*xi(2) - 1)*(xi(1) + 1))/2;
    d2Ndxi1dxi3 = -(xi(2)*xi(3)*(2*xi(1) + 1)*(xi(2) - 1))/2;
    d2Ndxi2dxi3 = -(xi(1)*xi(3)*(2*xi(2) - 1)*(xi(1) + 1))/2;
    d2Ndxi3dxi3 = -(xi(1)*xi(2)*(xi(1) + 1)*(xi(2) - 1))/2;

  case 15
    d2Ndxi1dxi1 = -(xi(2)*(xi(3)^2 - 1)*(xi(2) + 1))/2;
    d2Ndxi2dxi1 = -((2*xi(1) + 1)*(2*xi(2) + 1)*(xi(3)^2 - 1))/4;
    d2Ndxi3dxi1 = -(xi(2)*xi(3)*(2*xi(1) + 1)*(xi(2) + 1))/2;
    d2Ndxi1dxi2 = -((2*xi(1) + 1)*(2*xi(2) + 1)*(xi(3)^2 - 1))/4;
    d2Ndxi2dxi2 = -(xi(1)*(xi(3)^2 - 1)*(xi(1) + 1))/2;
    d2Ndxi3dxi2 = -(xi(1)*xi(3)*(2*xi(2) + 1)*(xi(1) + 1))/2;
    d2Ndxi1dxi3 = -(xi(2)*xi(3)*(2*xi(1) + 1)*(xi(2) + 1))/2;
    d2Ndxi2dxi3 = -(xi(1)*xi(3)*(2*xi(2) + 1)*(xi(1) + 1))/2;
    d2Ndxi3dxi3 = -(xi(1)*xi(2)*(xi(1) + 1)*(xi(2) + 1))/2;

  case 16
    d2Ndxi1dxi1 = -(xi(2)*(xi(3)^2 - 1)*(xi(2) + 1))/2;
    d2Ndxi2dxi1 = -((2*xi(1) - 1)*(2*xi(2) + 1)*(xi(3)^2 - 1))/4;
    d2Ndxi3dxi1 = -(xi(2)*xi(3)*(2*xi(1) - 1)*(xi(2) + 1))/2;
    d2Ndxi1dxi2 = -((2*xi(1) - 1)*(2*xi(2) + 1)*(xi(3)^2 - 1))/4;
    d2Ndxi2dxi2 = -(xi(1)*(xi(3)^2 - 1)*(xi(1) - 1))/2;
    d2Ndxi3dxi2 = -(xi(1)*xi(3)*(2*xi(2) + 1)*(xi(1) - 1))/2;
    d2Ndxi1dxi3 = -(xi(2)*xi(3)*(2*xi(1) - 1)*(xi(2) + 1))/2;
    d2Ndxi2dxi3 = -(xi(1)*xi(3)*(2*xi(2) + 1)*(xi(1) - 1))/2;
    d2Ndxi3dxi3 = -(xi(1)*xi(2)*(xi(1) - 1)*(xi(2) + 1))/2;

  case 17
    d2Ndxi1dxi1 = -(xi(2)*xi(3)*(xi(2) - 1)*(xi(3) + 1))/2;
    d2Ndxi2dxi1 = -(xi(1)*xi(3)*(2*xi(2) - 1)*(xi(3) + 1))/2;
    d2Ndxi3dxi1 = -(xi(1)*xi(2)*(2*xi(3) + 1)*(xi(2) - 1))/2;
    d2Ndxi1dxi2 = -(xi(1)*xi(3)*(2*xi(2) - 1)*(xi(3) + 1))/2;
    d2Ndxi2dxi2 = -(xi(3)*(xi(1)^2 - 1)*(xi(3) + 1))/2;
    d2Ndxi3dxi2 = -((2*xi(2) - 1)*(2*xi(3) + 1)*(xi(1)^2 - 1))/4;
    d2Ndxi1dxi3 = -(xi(1)*xi(2)*(2*xi(3) + 1)*(xi(2) - 1))/2;
    d2Ndxi2dxi3 = -((2*xi(2) - 1)*(2*xi(3) + 1)*(xi(1)^2 - 1))/4;
    d2Ndxi3dxi3 = -(xi(2)*(xi(1)^2 - 1)*(xi(2) - 1))/2;

  case 18
    d2Ndxi1dxi1 = -(xi(3)*(xi(2)^2 - 1)*(xi(3) + 1))/2;
    d2Ndxi2dxi1 = -(xi(2)*xi(3)*(2*xi(1) + 1)*(xi(3) + 1))/2;
    d2Ndxi3dxi1 = -((2*xi(1) + 1)*(2*xi(3) + 1)*(xi(2)^2 - 1))/4;
    d2Ndxi1dxi2 = -(xi(2)*xi(3)*(2*xi(1) + 1)*(xi(3) + 1))/2;
    d2Ndxi2dxi2 = -(xi(1)*xi(3)*(xi(1) + 1)*(xi(3) + 1))/2;
    d2Ndxi3dxi2 = -(xi(1)*xi(2)*(2*xi(3) + 1)*(xi(1) + 1))/2;
    d2Ndxi1dxi3 = -((2*xi(1) + 1)*(2*xi(3) + 1)*(xi(2)^2 - 1))/4;
    d2Ndxi2dxi3 = -(xi(1)*xi(2)*(2*xi(3) + 1)*(xi(1) + 1))/2;
    d2Ndxi3dxi3 = -(xi(1)*(xi(2)^2 - 1)*(xi(1) + 1))/2;

  case 19
```

```
    d2Ndxi1dxi1 = -(xi(2)*xi(3)*(xi(2) + 1)*(xi(3) + 1))/2;
    d2Ndxi2dxi1 = -(xi(1)*xi(3)*(2*xi(2) + 1)*(xi(3) + 1))/2;
    d2Ndxi3dxi1 = -(xi(1)*xi(2)*(2*xi(3) + 1)*(xi(2) + 1))/2;
    d2Ndxi1dxi2 = -(xi(1)*xi(3)*(2*xi(2) + 1)*(xi(3) + 1))/2;
    d2Ndxi2dxi2 = -(xi(3)*(xi(1)^2 - 1)*(xi(3) + 1))/2;
    d2Ndxi3dxi2 = -((2*xi(2) + 1)*(2*xi(3) + 1)*(xi(1)^2 - 1))/4;
    d2Ndxi1dxi3 = -(xi(1)*xi(2)*(2*xi(3) + 1)*(xi(2) + 1))/2;
    d2Ndxi2dxi3 = -((2*xi(2) + 1)*(2*xi(3) + 1)*(xi(1)^2 - 1))/4;
    d2Ndxi3dxi3 = -(xi(2)*(xi(1)^2 - 1)*(xi(2) + 1))/2;

  case 20
    d2Ndxi1dxi1 = -(xi(3)*(xi(2)^2 - 1)*(xi(3) + 1))/2;
    d2Ndxi2dxi1 = -(xi(2)*xi(3)*(2*xi(1) - 1)*(xi(3) + 1))/2;
    d2Ndxi3dxi1 = -((2*xi(1) - 1)*(2*xi(3) + 1)*(xi(2)^2 - 1))/4;
    d2Ndxi1dxi2 = -(xi(2)*xi(3)*(2*xi(1) - 1)*(xi(3) + 1))/2;
    d2Ndxi2dxi2 = -(xi(1)*xi(3)*(xi(1) - 1)*(xi(3) + 1))/2;
    d2Ndxi3dxi2 = -(xi(1)*xi(2)*(2*xi(3) + 1)*(xi(1) - 1))/2;
    d2Ndxi1dxi3 = -((2*xi(1) - 1)*(2*xi(3) + 1)*(xi(2)^2 - 1))/4;
    d2Ndxi2dxi3 = -(xi(1)*xi(2)*(2*xi(3) + 1)*(xi(1) - 1))/2;
    d2Ndxi3dxi3 = -(xi(1)*(xi(2)^2 - 1)*(xi(1) - 1))/2;

  case 21
    d2Ndxi1dxi1 = xi(3)*(xi(2)^2 - 1)*(xi(3) - 1);
    d2Ndxi2dxi1 = 2*xi(1)*xi(2)*xi(3)*(xi(3) - 1);
    d2Ndxi3dxi1 = xi(1)*(2*xi(3) - 1)*(xi(2)^2 - 1);
    d2Ndxi1dxi2 = 2*xi(1)*xi(2)*xi(3)*(xi(3) - 1);
    d2Ndxi2dxi2 = xi(3)*(xi(1)^2 - 1)*(xi(3) - 1);
    d2Ndxi3dxi2 = xi(2)*(2*xi(3) - 1)*(xi(1)^2 - 1);
    d2Ndxi1dxi3 = xi(1)*(2*xi(3) - 1)*(xi(2)^2 - 1);
    d2Ndxi2dxi3 = xi(2)*(2*xi(3) - 1)*(xi(1)^2 - 1);
    d2Ndxi3dxi3 = (xi(1)^2 - 1)*(xi(2)^2 - 1);

  case 22
    d2Ndxi1dxi1 = xi(2)*(xi(3)^2 - 1)*(xi(2) - 1);
    d2Ndxi2dxi1 = xi(1)*(2*xi(2) - 1)*(xi(3)^2 - 1);
    d2Ndxi3dxi1 = 2*xi(1)*xi(2)*xi(3)*(xi(2) - 1);
    d2Ndxi1dxi2 = xi(1)*(2*xi(2) - 1)*(xi(3)^2 - 1);
    d2Ndxi2dxi2 = (xi(1)^2 - 1)*(xi(3)^2 - 1);
    d2Ndxi3dxi2 = xi(3)*(2*xi(2) - 1)*(xi(1)^2 - 1);
    d2Ndxi1dxi3 = 2*xi(1)*xi(2)*xi(3)*(xi(2) - 1);
    d2Ndxi2dxi3 = xi(3)*(2*xi(2) - 1)*(xi(1)^2 - 1);
    d2Ndxi3dxi3 = xi(2)*(xi(1)^2 - 1)*(xi(2) - 1);

  case 23
    d2Ndxi1dxi1 = (xi(2)^2 - 1)*(xi(3)^2 - 1);
    d2Ndxi2dxi1 = xi(2)*(2*xi(1) + 1)*(xi(3)^2 - 1);
    d2Ndxi3dxi1 = xi(3)*(2*xi(1) + 1)*(xi(2)^2 - 1);
    d2Ndxi1dxi2 = xi(2)*(2*xi(1) + 1)*(xi(3)^2 - 1);
    d2Ndxi2dxi2 = xi(1)*(xi(3)^2 - 1)*(xi(1) + 1);
    d2Ndxi3dxi2 = 2*xi(1)*xi(2)*xi(3)*(xi(1) + 1);
    d2Ndxi1dxi3 = xi(3)*(2*xi(1) + 1)*(xi(2)^2 - 1);
    d2Ndxi2dxi3 = 2*xi(1)*xi(2)*xi(3)*(xi(1) + 1);
```

```
          d2Ndxi3dxi3 = xi(1)*(xi(2)^2 - 1)*(xi(1) + 1);

        case 24
          d2Ndxi1dxi1 = xi(2)*(xi(3)^2 - 1)*(xi(2) + 1);
          d2Ndxi2dxi1 = xi(1)*(2*xi(2) + 1)*(xi(3)^2 - 1);
          d2Ndxi3dxi1 = 2*xi(1)*xi(2)*xi(3)*(xi(2) + 1);
          d2Ndxi1dxi2 = xi(1)*(2*xi(2) + 1)*(xi(3)^2 - 1);
          d2Ndxi2dxi2 = (xi(1)^2 - 1)*(xi(3)^2 - 1);
          d2Ndxi3dxi2 = xi(3)*(2*xi(2) + 1)*(xi(1)^2 - 1);
          d2Ndxi1dxi3 = 2*xi(1)*xi(2)*xi(3)*(xi(2) + 1);
          d2Ndxi2dxi3 = xi(3)*(2*xi(2) + 1)*(xi(1)^2 - 1);
          d2Ndxi3dxi3 = xi(2)*(xi(1)^2 - 1)*(xi(2) + 1);

        case 25
          d2Ndxi1dxi1 = (xi(2)^2 - 1)*(xi(3)^2 - 1);
          d2Ndxi2dxi1 = xi(2)*(2*xi(1) - 1)*(xi(3)^2 - 1);
          d2Ndxi3dxi1 = xi(3)*(2*xi(1) - 1)*(xi(2)^2 - 1);
          d2Ndxi1dxi2 = xi(2)*(2*xi(1) - 1)*(xi(3)^2 - 1);
          d2Ndxi2dxi2 = xi(1)*(xi(3)^2 - 1)*(xi(1) - 1);
          d2Ndxi3dxi2 = 2*xi(1)*xi(2)*xi(3)*(xi(1) - 1);
          d2Ndxi1dxi3 = xi(3)*(2*xi(1) - 1)*(xi(2)^2 - 1);
          d2Ndxi2dxi3 = 2*xi(1)*xi(2)*xi(3)*(xi(1) - 1);
          d2Ndxi3dxi3 = xi(1)*(xi(2)^2 - 1)*(xi(1) - 1);

        case 26
          d2Ndxi1dxi1 = xi(3)*(xi(2)^2 - 1)*(xi(3) + 1);
          d2Ndxi2dxi1 = 2*xi(1)*xi(2)*xi(3)*(xi(3) + 1);
          d2Ndxi3dxi1 = xi(1)*(2*xi(3) + 1)*(xi(2)^2 - 1);
          d2Ndxi1dxi2 = 2*xi(1)*xi(2)*xi(3)*(xi(3) + 1);
          d2Ndxi2dxi2 = xi(3)*(xi(1)^2 - 1)*(xi(3) + 1);
          d2Ndxi3dxi2 = xi(2)*(2*xi(3) + 1)*(xi(1)^2 - 1);
          d2Ndxi1dxi3 = xi(1)*(2*xi(3) + 1)*(xi(2)^2 - 1);
          d2Ndxi2dxi3 = xi(2)*(2*xi(3) + 1)*(xi(1)^2 - 1);
          d2Ndxi3dxi3 = (xi(1)^2 - 1)*(xi(2)^2 - 1);

        case 27
          d2Ndxi1dxi1 = -2*(xi(2)^2 - 1)*(xi(3)^2 - 1);
          d2Ndxi2dxi1 = -4*xi(1)*xi(2)*(xi(3)^2 - 1);
          d2Ndxi3dxi1 = -4*xi(1)*xi(3)*(xi(2)^2 - 1);
          d2Ndxi1dxi2 = -4*xi(1)*xi(2)*(xi(3)^2 - 1);
          d2Ndxi2dxi2 = -2*(xi(1)^2 - 1)*(xi(3)^2 - 1);
          d2Ndxi3dxi2 = -4*xi(2)*xi(3)*(xi(1)^2 - 1);
          d2Ndxi1dxi3 = -4*xi(1)*xi(3)*(xi(2)^2 - 1);
          d2Ndxi2dxi3 = -4*xi(2)*xi(3)*(xi(1)^2 - 1);
          d2Ndxi3dxi3 = -2*(xi(1)^2 - 1)*(xi(2)^2 - 1);

      end

      switch( i_eval )
        case 22
          vBase = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
```

```
                  aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3
        case 33
          vBase = aInvJac(:,4).*( aInvJac(:,4)*d2Ndxi1dxi1 + aInvJac(:,5)*d2Ndxi2dxi1 + aInvJac(:,6
                  aInvJac(:,5).*( aInvJac(:,4)*d2Ndxi1dxi2 + aInvJac(:,5)*d2Ndxi2dxi2 + aInvJac(:,6
                  aInvJac(:,6).*( aInvJac(:,4)*d2Ndxi1dxi3 + aInvJac(:,5)*d2Ndxi2dxi3 + aInvJac(:,6

        case 44
          vBase = aInvJac(:,7).*( aInvJac(:,7)*d2Ndxi1dxi1 + aInvJac(:,8)*d2Ndxi2dxi1 + aInvJac(:,9
                  aInvJac(:,8).*( aInvJac(:,7)*d2Ndxi1dxi2 + aInvJac(:,8)*d2Ndxi2dxi2 + aInvJac(:,9
                  aInvJac(:,9).*( aInvJac(:,7)*d2Ndxi1dxi3 + aInvJac(:,8)*d2Ndxi2dxi3 + aInvJac(:,9

        case {23,32}
          vBase = aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,5).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
                  aInvJac(:,6).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3

        case {24,42}
          vBase = aInvJac(:,7).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:,2)*d2Ndxi2dxi1 + aInvJac(:,3
                  aInvJac(:,8).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:,2)*d2Ndxi2dxi2 + aInvJac(:,3
                  aInvJac(:,9).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:,2)*d2Ndxi2dxi3 + aInvJac(:,3

        case {34,43}
          vBase = aInvJac(:,7).*( aInvJac(:,4)*d2Ndxi1dxi1 + aInvJac(:,5)*d2Ndxi2dxi1 + aInvJac(:,6
                  aInvJac(:,8).*( aInvJac(:,4)*d2Ndxi1dxi2 + aInvJac(:,5)*d2Ndxi2dxi2 + aInvJac(:,6
                  aInvJac(:,9).*( aInvJac(:,4)*d2Ndxi1dxi3 + aInvJac(:,5)*d2Ndxi2dxi3 + aInvJac(:,6
      end

    otherwise
      vBase = 0;

  end
```

11.46 ellib/sf_hex_Q3

SF_HEX_Q3 Tricubic conforming shape function for hexahedrons (Q3).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_HEX_Q3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming tricubic Q3 shape functions on hexahedrons with values defined in the nodes, edges, faces, and cell center. XI is
[-1..1]^3 reference coordinates.

```
Input       Value/[Size]           Description
--------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  3             Number of space dimensions
```

```
n_vert      scalar:  8              Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                                .
Output      Value/[Size]           Description
--------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag3, sf_hex_Q1

Code listing

```
 nLDof = [8 24 24 8];
 xLDof = [ -1  1  1 -1 -1  1  1 -1  -1/3 1   1/3 -1   -1    1    1   -1  -1/3 1   1/3 -1
1/3 1   -1/3 -1   -1   1   1  -1   1/3 1  -1/3 -1   -1/3 -1/3  1    1/3 -1   -1/3
1/3  1/3  1   -1/3 -1    1/3  1/3  1/3 1   -1/3 -1   1/3 -1/3 -1/3  1   1/3 -1  -1/3
-1/3  1/3  1/3 -1/3 -1/3  1/3  1/3 -1/3 ;
            -1 -1  1  1 -1 -1  1  1  -1  -1/3 1    1/3 -1   -1    1    1  -1  -1/3 1
1/3 -1   1/3  1   -1/3 -1  -1   1   1  -1   1/3 1   -1/3 -1/3 -1   -1/3  1    1/3 -1/3 -1/3 -1
1/3  1   -1/3 -1/3  1/3 -1   1/3  1   -1/3 1/3  1/3 -1   -1/3 1    1/3 1/3   -1/3 -1/3
1/3  1/3 -1/3 -1/3  1/3  1/3 ;
           -1 -1 -1 -1  1  1  1  1  -1  -1  -1   -1   -1/3 -1/3 -1/3 -1/3 1   1   1
1   -1  -1   -1   -1    1/3 1/3 1/3 1/3 1   1   1    1   -1   -1/3 -1/3 -1/3 -1/3
1   -1   -1/3 -1/3 -1/3 -1/3  1   -1    1/3 1/3  1/3  1/3 1   -1    1/3  1/3 1/3  1/3 1
-1/3 -1/3 -1/3 -1/3  1/3  1/3  1/3  1/3 ];
 sfun  = 'sf_hex_Q3';


 switch i_eval

   case 1

     switch i_dof
       case 1
         vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
       case 2
         vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
```

```
        case 3
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 4
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 5
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 6
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 7
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 8
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 9
          vBase = (3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 10
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 11
          vBase = (3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 12
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 13
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(9*xi(3) +
        case 14
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(9*xi(3) +
        case 15
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(9*xi(3) +
        case 16
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(9*xi(3) +
        case 17
          vBase = -(3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 18
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 19
          vBase = -(3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 20
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 21
          vBase = -(3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 22
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 23
          vBase = -(3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 24
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 25
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(9*xi(3) +
        case 26
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(9*xi(3) +
        case 27
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(9*xi(3) +
        case 28
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(9*xi(3) +
```

```
        case 29
          vBase = (3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 30
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 31
          vBase = (3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) +
        case 32
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 33
          vBase = -(3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 34
          vBase = -(3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(9*xi(3) +
        case 35
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(9*xi(3) +
        case 36
          vBase = -(3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(9*xi(3) +
        case 37
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(9*xi(3) +
        case 38
          vBase = (3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 39
          vBase = (3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 40
          vBase = (3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(9*xi(3) +
        case 41
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(9*xi(3) +
        case 42
          vBase = (3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(9*xi(3) +
        case 43
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(9*xi(3) +
        case 44
          vBase = -(3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 45
          vBase = -(3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 46
          vBase = -(3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(9*xi(3) +
        case 47
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) + 1)*(9*xi(3) +
        case 48
          vBase = -(3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(9*xi(3) +
        case 49
          vBase = (3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) + 1)*(9*xi(3) +
        case 50
          vBase = (3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 51
          vBase = (3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 52
          vBase = (3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(9*xi(3) +
        case 53
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) + 1)*(9*xi(3) +
        case 54
          vBase = (3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(9*xi(3) +
```

```
        case 55
          vBase = -(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) + 1)*(9*xi(3) +
        case 56
          vBase = -(3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(3*xi(3) +
        case 57
          vBase = (3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(9*xi(3) +
        case 58
          vBase = -(3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(9*xi(3) +
        case 59
          vBase = (3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(9*xi(3) +
        case 60
          vBase = -(3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) - 1)*(9*xi(3) +
        case 61
          vBase = -(3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) + 1)*(9*xi(3) +
        case 62
          vBase = (3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) - 1)*(9*xi(2) + 9)*(3*xi(3) + 1)*(9*xi(3) +
        case 63
          vBase = -(3*xi(1) + 1)*(9*xi(1) + 9)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) + 1)*(9*xi(3) +
        case 64
          vBase = (3*xi(1) - 1)*(9*xi(1) + 9)*(3*xi(2) + 1)*(9*xi(2) + 9)*(3*xi(3) + 1)*(9*xi(3) +
      end

    case {2,3,4}

      switch i_dof
        case 1
          dNdxi1 = ((3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) - 1)
          dNdxi2 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) - 1)
          dNdxi3 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) - 1)
        case 2
          dNdxi1 = ((3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) - 1)
          dNdxi2 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) - 1
          dNdxi3 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) - 1
        case 3
          dNdxi1 = -((3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) - 1
          dNdxi2 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) - 1
          dNdxi3 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) + 1)
        case 4
          dNdxi1 = -((3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) - 1
          dNdxi2 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) - 1)
          dNdxi3 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) + 1
        case 5
          dNdxi1 = -((3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) + 1
          dNdxi2 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) + 1
          dNdxi3 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) - 1)
        case 6
          dNdxi1 = -((3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) + 1
          dNdxi2 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) + 1)
          dNdxi3 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) - 1
        case 7
          dNdxi1 = ((3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) + 1)
          dNdxi2 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) + 1)
```

```
          dNdxi3 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) + 1)
        case 8
          dNdxi1 = ((3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) + 1)
          dNdxi2 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) + 1
          dNdxi3 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) + 1
        case 9
          dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) -
          dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1
          dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1
        case 10
          dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1
          dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) -
          dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
        case 11
          dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) -
          dNdxi2 = (9*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)
          dNdxi3 = -(9*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) + 1
        case 12
          dNdxi1 = (9*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)
          dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) -
          dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) - 1)*(xi(2) + 1)
        case 13
          dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) -
        case 14
          dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) -
        case 15
          dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) +
        case 16
          dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) +
        case 17
          dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) +
          dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) + 1)
          dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1
        case 18
          dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) + 1)
          dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) +
          dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
        case 19
          dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) +
          dNdxi2 = -(9*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) + 1
          dNdxi3 = -(9*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) + 1
        case 20
          dNdxi1 = -(9*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) + 1
          dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) +
```

```
      dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) - 1)*(xi(2) + 1)
    case 21
      dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) -
      dNdxi2 = (9*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)
      dNdxi3 = (9*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)
    case 22
      dNdxi1 = (9*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)
      dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) -
      dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1
    case 23
      dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) -
      dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1
      dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) + 1)
    case 24
      dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1
      dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) -
      dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(2) - 1)*(xi(2) + 1
    case 25
      dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) - 1)*(xi(3) + 1)
      dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) - 1)*(xi(3) + 1)
      dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) -
    case 26
      dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) - 1)*(xi(3) + 1)
      dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1
      dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) -
    case 27
      dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1
      dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1
      dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) +
    case 28
      dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1
      dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) - 1)*(xi(3) + 1)
      dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) +
    case 29
      dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) +
      dNdxi2 = -(9*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) + 1
      dNdxi3 = (9*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)
    case 30
      dNdxi1 = -(9*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) + 1
      dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) +
      dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1
    case 31
      dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) +
      dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) + 1)
      dNdxi3 = (9*(3*xi(1) - 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) + 1)
    case 32
      dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) + 1)
      dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) +
      dNdxi3 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(2) - 1)*(xi(2) + 1
    case 33
      dNdxi1 = (81*(3*xi(2) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1
      dNdxi2 = (81*(3*xi(1) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1
```

```
        dNdxi3 = (81*(3*xi(1) - 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)*
      case 34
        dNdxi1 = (81*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi2 = (81*(3*xi(1) - 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)*
        dNdxi3 = (81*(3*xi(1) - 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1
      case 35
        dNdxi1 = (81*(3*xi(2) - 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)*
        dNdxi2 = -(81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) +
        dNdxi3 = -(81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) +
      case 36
        dNdxi1 = -(81*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) +
        dNdxi2 = -(81*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi3 = (81*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) + 1
      case 37
        dNdxi1 = -(81*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi2 = (81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi3 = -(81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) - 1)*(xi(2) +
      case 38
        dNdxi1 = -(81*(3*xi(2) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) +
        dNdxi2 = -(81*(3*xi(1) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) +
        dNdxi3 = (81*(3*xi(1) - 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)*
      case 39
        dNdxi1 = (81*(3*xi(2) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1
        dNdxi2 = -(81*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) -
        dNdxi3 = -(81*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
      case 40
        dNdxi1 = (81*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi2 = -(81*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi3 = -(81*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) -
      case 41
        dNdxi1 = -(81*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi2 = -(81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) +
        dNdxi3 = (81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1
      case 42
        dNdxi1 = -(81*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) +
        dNdxi2 = (81*(3*xi(1) - 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)*
        dNdxi3 = -(81*(3*xi(1) - 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) +
      case 43
        dNdxi1 = (81*(3*xi(2) - 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)*
        dNdxi2 = (81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi3 = (81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(2) - 1)*(xi(2) + 1
      case 44
        dNdxi1 = -(81*(3*xi(2) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) +
        dNdxi2 = (81*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) + 1
        dNdxi3 = -(81*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
      case 45
        dNdxi1 = -(81*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) -
        dNdxi2 = -(81*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) -
        dNdxi3 = (81*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)*
      case 46
        dNdxi1 = -(81*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) - 1)*(xi(3) +
        dNdxi2 = (81*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)*
```

```
        dNdxi3 = -(81*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) -
      case 47
        dNdxi1 = (81*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)*
        dNdxi2 = (81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi3 = (81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1
      case 48
        dNdxi1 = (81*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi2 = -(81*(3*xi(1) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi3 = -(81*(3*xi(1) - 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) +
      case 49
        dNdxi1 = -(81*(3*xi(2) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi2 = -(81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) - 1)*(xi(3) +
        dNdxi3 = (81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(2) - 1)*(xi(2) + 1
      case 50
        dNdxi1 = (81*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) + 1
        dNdxi2 = (81*(3*xi(1) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) + 1
        dNdxi3 = (81*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)*
      case 51
        dNdxi1 = -(81*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) -
        dNdxi2 = (81*(3*xi(1) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1
        dNdxi3 = -(81*(3*xi(1) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
      case 52
        dNdxi1 = -(81*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(3) - 1)*(xi(3) +
        dNdxi2 = -(81*(3*xi(1) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi3 = (81*(3*xi(1) - 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1
      case 53
        dNdxi1 = -(81*(3*xi(2) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi2 = (81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi3 = -(81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) +
      case 54
        dNdxi1 = (81*(3*xi(2) - 1)*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi2 = (81*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)*
        dNdxi3 = (81*(3*xi(1) + 1)*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) + 1
      case 55
        dNdxi1 = (81*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)*
        dNdxi2 = -(81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(3) - 1)*(xi(3) +
        dNdxi3 = -(81*(3*xi(1) - 1)*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(2) - 1)*(xi(2) +
      case 56
        dNdxi1 = (81*(3*xi(2) + 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) + 1
        dNdxi2 = -(81*(3*xi(1) - 1)*(3*xi(3) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) +
        dNdxi3 = -(81*(3*xi(1) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
      case 57
        dNdxi1 = -(729*(3*xi(2) - 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi2 = -(729*(3*xi(1) - 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi3 = -(729*(3*xi(1) - 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1
      case 58
        dNdxi1 = -(729*(3*xi(2) - 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1
        dNdxi2 = (729*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi3 = (729*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
      case 59
        dNdxi1 = (729*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
        dNdxi2 = (729*(3*xi(1) + 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
```

```
          dNdxi3 = -(729*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1
        case 60
          dNdxi1 = (729*(3*xi(2) + 1)*(3*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi2 = -(729*(3*xi(1) - 1)*(3*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi3 = (729*(3*xi(1) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
        case 61
          dNdxi1 = (729*(3*xi(2) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi2 = (729*(3*xi(1) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi3 = -(729*(3*xi(1) - 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1
        case 62
          dNdxi1 = (729*(3*xi(2) - 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi2 = -(729*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi3 = (729*(3*xi(1) + 1)*(3*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
        case 63
          dNdxi1 = -(729*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi2 = -(729*(3*xi(1) + 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi3 = -(729*(3*xi(1) + 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1
        case 64
          dNdxi1 = -(729*(3*xi(2) + 1)*(3*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*(xi(3) + 1
          dNdxi2 = (729*(3*xi(1) - 1)*(3*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*(xi(3) + 1)
          dNdxi3 = (729*(3*xi(1) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1)
      end

      if     ( i_eval==2 )   % x-derivative.

        vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3;

      elseif ( i_eval==3 )   % y-derivative.

        vBase = aInvJac(:,4)*dNdxi1 + aInvJac(:,5)*dNdxi2 + aInvJac(:,6)*dNdxi3;

      elseif ( i_eval==4 )   % z-derivative.

        vBase = aInvJac(:,7)*dNdxi1 + aInvJac(:,8)*dNdxi2 + aInvJac(:,9)*dNdxi3;
      end

    case {22,23,24,32,33,34,42,43,44}   % Evaluation of second order derivatives.
      error('sf_hex_Q3: second order derivative evaluation not supported.')

    otherwise
      vBase = 0;

  end
```

11.47 ellib/sf_hex_Q4

SF_HEX_Q4 Triquartic conforming shape function for hexahedrons (Q4).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_HEX_Q4(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming triquartic Q4 shape functions on hexahedrons with values defined in the nodes, edges, faces, and cell center. XI is
[-1..1]^3 reference coordinates.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  3             Number of space dimensions
n_vert      scalar:  8             Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                            .
Output      Value/[Size]           Description
-----------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag4, sf_hex_Q1

Code listing

```
 nLDof = [8 36 54 27];
 xLDof = [-1  1  1 -1 -1  1 1 -1 -1/2    1 1/2  -1   -1    1    1   -1 -1/2    1 1/2
-1  0  1  0 -1 -1  1 1 -1  0 1 0 -1 1/2   1 -1/2   -1  -1   1   1  -1 1/2   1 -1/2
-1 -1/2 -1/2    1  1/2   -1 -1/2    0    0    1    0   -1    0  1/2  1/2    1 -1/2
-1  1/2 1/2 1/2   1 -1/2   -1 1/2 1/2 1/2   1 -1/2   -1 1/2   0   0   1   0  -1   0 -1/2 -1/2
1 1/2  -1 -1/2 -1/2 -1/2    1 1/2  -1 -1/2  0  0 1 0 -1 0 -1/2    0  1/2 -1/2    0
1/2 -1/2    0  1/2 -1/2    0  1/2 -1/2 0 1/2 -1/2   0 1/2 -1/2    0  1/2 -1/2   0 1/2 -1/2
0 1/2;
          -1 -1  1  1 -1 -1 1  1   -1 -1/2   1 1/2   -1   -1    1    1   -1 -1/2
1 1/2 -1  0  1  0 -1 -1 1  1 -1 0 1  0  -1 1/2    1 -1/2  -1  -1   1   1  -1 1/2
1 -1/2 -1/2   -1 -1/2    1  1/2 -1/2 -1/2   -1    0    1    0 -1/2 -1/2   -1  1/2
1 -1/2 -1/2   0  -1 1/2    1 -1/2   0 1/2  -1 1/2    1 -1/2 1/2 1/2  -1   0   1   0 1/2
1/2   -1 -1/2   1 1/2  1/2    0   -1 -1/2   1 1/2    0  0 -1 0 1  0 0 -1/2 -1/2 -1/2
```

```
0    0    0  1/2  1/2  1/2 -1/2 -1/2 -1/2    0 0   0  1/2 1/2 1/2 -1/2 -1/2 -1/2
0   0   0  1/2 1/2 1/2;
          -1 -1 -1 -1  1  1 1  1   -1   -1  -1 -1/2 -1/2 -1/2 -1/2    1    1
1   1 -1 -1 -1 -1  0  0 0  0  1 1 1  1  -1  -1   -1   -1 1/2 1/2 1/2 1/2   1   1
1    1   -1 -1/2 -1/2 -1/2 -1/2    1   -1 -1/2 -1/2 -1/2 -1/2    1   -1 -1/2 -1/2 -1/2 -1/2
1  -1   0   0    0    0   1  -1 1/2 1/2  1/2  1/2   1  -1 1/2 1/2 1/2 1/2   1   -1
1/2  1/2 1/2 1/2    1   -1    0    0   0   0    1 -1  0 0 0  0 1 -1/2 -1/2 -1/2 -1/2 -1/2 -1/2 -1/
0    0    0    0 0   0    0   0   0  1/2  1/2  1/2  1/2 1/2 1/2  1/2 1/2 1/2];
 sfun  = 'sf_hex_Q4';


 switch i_eval

   case 1

     switch i_dof
       case 1
         vBase = -(xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^
       case 2
         vBase = -(xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 + 4*xi(2)^
       case 3
         vBase = -(xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 - 4*xi(2)^
       case 4
         vBase = -(xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2)^
       case 5
         vBase = -(xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^
       case 6
         vBase = -(xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 + 4*xi(2)^
       case 7
         vBase = -(xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 - 4*xi(2)^
       case 8
         vBase = -(xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2)^
       case 9
         vBase = (xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^2
       case 10
         vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 2*xi(2)^3 + xi(2)^2 +
       case 11
         vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)*(- 4*xi(3)^3 + 4*xi(3)^2
       case 12
         vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(3)^3 + 4*xi(3)^2
       case 13
         vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^2
       case 14
         vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 + 4*xi(2)^2
       case 15
         vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 - 4*xi(2)^2
       case 16
         vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2)^2
       case 17
         vBase = (xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^2
       case 18
         vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 2*xi(2)^3 + xi(2)^2 +
```

```
        case 19
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)*(- 4*xi(3)^3 - 4*xi(3)^2
        case 20
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(3)^3 - 4*xi(3)^2
        case 21
          vBase = (xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)*(-
        case 22
          vBase = (xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(-
        case 23
          vBase = (xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)*(-
        case 24
          vBase = (xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(-
        case 25
          vBase = (xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(-
        case 26
          vBase = (xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(-
        case 27
          vBase = (xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(-
        case 28
          vBase = (xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(-
        case 29
          vBase = (xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)*(-
        case 30
          vBase = (xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(-
        case 31
          vBase = (xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)*(-
        case 32
          vBase = (xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(-
        case 33
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)*(- 4*xi(3)^3 + 4*xi(3)^2
        case 34
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(3)^3 + 4*xi(3)^2
        case 35
          vBase = (xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2)^2
        case 36
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^2 +
        case 37
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^2
        case 38
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 + 4*xi(2)^2
        case 39
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 - 4*xi(2)^2
        case 40
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2)^2
        case 41
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)*(- 4*xi(3)^3 - 4*xi(3)^2
        case 42
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(3)^3 - 4*xi(3)^2
        case 43
          vBase = (xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2)^2
        case 44
          vBase = (xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^2 +
```

```
        case 45
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^
        case 46
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2
        case 47
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 2*xi(2)^3 + xi(2)^
        case 48
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)*(- 2*xi(3)^3 + xi(3)^
        case 49
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 2*xi(3)^3 + xi(3)^
        case 50
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^
        case 51
          vBase = -(2*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)
        case 52
          vBase = -(2*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)
        case 53
          vBase = -(2*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)
        case 54
          vBase = -(2*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)
        case 55
          vBase = -(2*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)
        case 56
          vBase = -(2*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)
        case 57
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)*(- 4*xi(3)^3 + 4*xi(3
        case 58
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)*(- 2*xi(3)^3 + xi(3)^
        case 59
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 2*xi(3)^3 + xi(3)^
        case 60
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2
        case 61
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^
        case 62
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)*(- 4*xi(3)^3 - 4*xi(3
        case 63
          vBase = -(2*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(3)^3 + 4*xi(3)^2 + xi(3) - 1)
        case 64
          vBase = -(2*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)
        case 65
          vBase = -(2*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)
        case 66
          vBase = -(2*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)
        case 67
          vBase = -(2*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)
        case 68
          vBase = -(2*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(3)^3 - 4*xi(3)^2 + xi(3) + 1)
        case 69
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(3)^3 + 4*xi(3)^2 + xi(3) - 1)*(- 2*xi(1)^3 - xi(1)^
        case 70
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)*(- 2*xi(1)^3 - xi(1)^
```

```
        case 71
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 2*xi(2)^3 - xi(2)^
        case 72
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2
        case 73
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^
        case 74
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(3)^3 - 4*xi(3)^2 + xi(3) + 1)*(- 2*xi(1)^3 - xi(1)^
        case 75
          vBase = -(2*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(3)^3 + 4*xi(3)^2 + xi(3) - 1)
        case 76
          vBase = -(2*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)
        case 77
          vBase = -(2*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)
        case 78
          vBase = -(2*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)
        case 79
          vBase = -(2*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)
        case 80
          vBase = -(2*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(3)^3 - 4*xi(3)^2 + xi(3) + 1)
        case 81
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(3)^3 + 4*xi(3
        case 82
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2
        case 83
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 2*xi(2)^3 + xi(2)^
        case 84
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)*(- 2*xi(1)^3 - xi(1)^
        case 85
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 2*xi(2)^3 - xi(2)^
        case 86
          vBase = -(8*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(3)^3 - 4*xi(3
        case 87
          vBase = -(2*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)
        case 88
          vBase = -(2*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)
        case 89
          vBase = -(2*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)
        case 90
          vBase = -(2*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)
        case 91
          vBase = -(2*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)
        case 92
          vBase = -(2*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)
        case 93
          vBase = -(xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(3)^3 + 4*
        case 94
          vBase = -(xi(2)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(2)^3 + 4*
        case 95
          vBase = -(xi(1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 - 4*
        case 96
          vBase = -(xi(2)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(2)^3 - 4*
```

```
        case 97
          vBase = -(xi(1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 4*xi(1)^3 + 4*
        case 98
          vBase = -(xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(3)^3 - 4*
        case 99
          vBase = (64*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^
        case 100
          vBase = (16*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)
        case 101
          vBase = (64*xi(1)*xi(2)*xi(3)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)*(- 2*xi(3)^3 + xi(3)^
        case 102
          vBase = (16*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)
        case 103
          vBase = (4*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(3)^3 + x
        case 104
          vBase = (16*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(3)^3 + xi(3)^2 + 2*xi(3) - 1)
        case 105
          vBase = (64*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 2*xi(3)^3 + xi(3)^
        case 106
          vBase = (16*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 2*xi(3)^3 + xi(3)^2 + 2*xi(3) - 1)
        case 107
          vBase = (64*xi(1)*xi(2)*xi(3)*(- 2*xi(3)^3 + xi(3)^2 + 2*xi(3) - 1)*(- 2*xi(1)^3 - xi(1)^
        case 108
          vBase = (16*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)
        case 109
          vBase = (4*xi(2)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(2)^3 + x
        case 110
          vBase = (16*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)
        case 111
          vBase = (4*xi(1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(1)^3 + x
        case 112
          vBase = (4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 +
        case 113
          vBase = (4*xi(1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(1)^3 - x
        case 114
          vBase = (16*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)
        case 115
          vBase = (4*xi(2)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(2)^3 - x
        case 116
          vBase = (16*xi(1)*xi(2)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 2*xi(1)^3 - xi(1)^2 + 2*xi(1) + 1)
        case 117
          vBase = (64*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^
        case 118
          vBase = (16*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)
        case 119
          vBase = (64*xi(1)*xi(2)*xi(3)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)*(- 2*xi(1)^3 - xi(1)^
        case 120
          vBase = (16*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)
        case 121
          vBase = (4*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(3)^3 - x
        case 122
          vBase = (16*xi(1)*xi(3)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(1)^3 - xi(1)^2 + 2*xi(1) + 1)
```

```
        case 123
          vBase = (64*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 2*xi(2)^3 - xi(2)^
        case 124
          vBase = (16*xi(2)*xi(3)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 2*xi(2)^3 - xi(2)^2 + 2*xi(2) + 1)
        case 125
          vBase = (64*xi(1)*xi(2)*xi(3)*(- 2*xi(1)^3 - xi(1)^2 + 2*xi(1) + 1)*(- 2*xi(2)^3 - xi(2)^
      end

    case {2,3,4}

      switch i_dof
        case 1
          dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) -
        case 2
          dNdxi1 = (xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) -
        case 3
          dNdxi1 = (xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) -
        case 4
          dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) -
        case 5
          dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) +
        case 6
          dNdxi1 = (xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) +
        case 7
          dNdxi1 = (xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) +
        case 8
          dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) +
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) +
        case 9
          dNdxi1 = (xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)
        case 10
          dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) + 1)
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(4*xi(3) - 1)*(xi(1) + 1)
        case 11
```

```
          dNdxi1 = (xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) + 1)
          dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)
        case 12
          dNdxi1 = (xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)
        case 13
          dNdxi1 = (xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) - 1)
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)
        case 14
          dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) + 1)
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) + 1)
        case 15
          dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) + 1
          dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) + 1
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) + 1)
        case 16
          dNdxi1 = (xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) + 1)
          dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) - 1
          dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)
        case 17
          dNdxi1 = (xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
          dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1
        case 18
          dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) + 1)
          dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(4*xi(3) + 1)*(xi(1) + 1
        case 19
          dNdxi1 = (xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) + 1)
          dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1
          dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1
        case 20
          dNdxi1 = (xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)
          dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
          dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1
        case 21
          dNdxi1 = (xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi
          dNdxi2 = -(xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(x
          dNdxi3 = -(xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(x
        case 22
          dNdxi1 = (xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi
          dNdxi2 = (xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi
          dNdxi3 = -(xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(x
        case 23
          dNdxi1 = (xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi
          dNdxi2 = (xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi
          dNdxi3 = -(xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(x
        case 24
```

```
      dNdxi1 = -(xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(x
      dNdxi2 = (xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi
      dNdxi3 = -(xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(x
    case 25
      dNdxi1 = -(xi(2)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(x
      dNdxi2 = -(xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(x
      dNdxi3 = (xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi
    case 26
      dNdxi1 = (xi(2)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi
      dNdxi2 = -(xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(x
      dNdxi3 = (xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi
    case 27
      dNdxi1 = (xi(2)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi
      dNdxi2 = (xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi
      dNdxi3 = (xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi
    case 28
      dNdxi1 = -(xi(2)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(x
      dNdxi2 = (xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi
      dNdxi3 = (xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi
    case 29
      dNdxi1 = (xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi
      dNdxi2 = -(xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(x
      dNdxi3 = (xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi
    case 30
      dNdxi1 = (xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi
      dNdxi2 = (xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi
      dNdxi3 = (xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi
    case 31
      dNdxi1 = (xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi
      dNdxi2 = (xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi
      dNdxi3 = (xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi
    case 32
      dNdxi1 = -(xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(x
      dNdxi2 = (xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi
      dNdxi3 = (xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi
    case 33
      dNdxi1 = (xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)
      dNdxi2 = (xi(1)*xi(3)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
      dNdxi3 = (xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)
    case 34
      dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1
      dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) + 1)
      dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) + 1)
    case 35
      dNdxi1 = (xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) + 1)
      dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1
      dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)
    case 36
      dNdxi1 = (xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)
      dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
      dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(4*xi(3) - 1)*(xi(1) - 1)
    case 37
```

```
        dNdxi1 = (xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)
        dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
        dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)
      case 38
        dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1
        dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) + 1)
        dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) + 1)
      case 39
        dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) + 1
        dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) + 1
        dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) + 1)
      case 40
        dNdxi1 = (xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) + 1)
        dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) - 1
        dNdxi3 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)
      case 41
        dNdxi1 = (xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)
        dNdxi2 = (xi(1)*xi(3)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
        dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1
      case 42
        dNdxi1 = -(xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1
        dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) + 1)
        dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) + 1
      case 43
        dNdxi1 = (xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) + 1)
        dNdxi2 = -(xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1
        dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1
      case 44
        dNdxi1 = (xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)
        dNdxi2 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)
        dNdxi3 = -(xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(4*xi(3) + 1)*(xi(1) - 1
      case 45
        dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(4*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1
      case 46
        dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(3) - 1
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1
      case 47
        dNdxi1 = (8*xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(1) + 1)*(xi(3) - 1
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(1) + 1)*(xi(2) - 1
      case 48
        dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) + 1)*(xi(3) - 1
        dNdxi2 = (8*xi(1)*xi(3)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1
      case 49
        dNdxi1 = -(8*xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(3) - 1
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(2) - 1
      case 50
```

```
      dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
      dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
      dNdxi3 = (8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(4*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)
    case 51
      dNdxi1 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*
      dNdxi2 = (2*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
      dNdxi3 = (2*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(4*xi(3) - 1)*(xi(1) - 1)*(xi
    case 52
      dNdxi1 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(8*xi(1)^2 - 5)*
      dNdxi2 = (2*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi
      dNdxi3 = (2*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi
    case 53
      dNdxi1 = -(2*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(x
      dNdxi2 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(8*xi(2)^2 - 5)*
      dNdxi3 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) + 1)*(xi
    case 54
      dNdxi1 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(8*xi(1)^2 - 5)*
      dNdxi2 = -(2*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(x
      dNdxi3 = (2*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi
    case 55
      dNdxi1 = (2*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi
      dNdxi2 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(8*xi(2)^2 - 5)*
      dNdxi3 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi
    case 56
      dNdxi1 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*
      dNdxi2 = (2*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
      dNdxi3 = -(2*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(4*xi(3) + 1)*(xi(1) - 1)*(x
    case 57
      dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
      dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
      dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(4*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1
    case 58
      dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(3) - 1
      dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1
      dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1
    case 59
      dNdxi1 = (8*xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)
      dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(1) + 1)*(xi(3) - 1
      dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(1) + 1)*(xi(2) - 1
    case 60
      dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) + 1)*(xi(3) - 1
      dNdxi2 = (8*xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)
      dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1
    case 61
      dNdxi1 = -(8*xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1
      dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(3) - 1
      dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(1) - 1)*(xi(2) - 1
    case 62
      dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
      dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
      dNdxi3 = (8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(4*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)
    case 63
```

```
          dNdxi1 = (2*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
          dNdxi2 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*
          dNdxi3 = (2*xi(1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)*(xi
        case 64
          dNdxi1 = (2*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
          dNdxi2 = (2*xi(1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
          dNdxi3 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*
        case 65
          dNdxi1 = -(2*xi(2)*(4*xi(1) + 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(x
          dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) + 1)*(xi
          dNdxi3 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*
        case 66
          dNdxi1 = (2*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) + 1)*(xi
          dNdxi2 = -(2*xi(1)*(2*xi(1) - 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(x
          dNdxi3 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*
        case 67
          dNdxi1 = (2*xi(2)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
          dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
          dNdxi3 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(8*xi(3)^2 - 5)*
        case 68
          dNdxi1 = (2*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
          dNdxi2 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*
          dNdxi3 = -(2*xi(1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1)*(x
        case 69
          dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
          dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
          dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1
        case 70
          dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(3) - 1
          dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
          dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1
        case 71
          dNdxi1 = (8*xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)
          dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(1) + 1)*(xi(3) - 1
          dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(1) + 1)*(xi(2) - 1
        case 72
          dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) + 1)*(xi(3) - 1
          dNdxi2 = (8*xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)
          dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1
        case 73
          dNdxi1 = -(8*xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
          dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(3) - 1
          dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(1) - 1)*(xi(2) - 1
        case 74
          dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
          dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
          dNdxi3 = (8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)
        case 75
          dNdxi1 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*
          dNdxi2 = (2*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
          dNdxi3 = (2*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)*(xi
        case 76
```

```
        dNdxi1 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*
        dNdxi2 = (2*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
        dNdxi3 = (2*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi
      case 77
        dNdxi1 = -(2*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(x
        dNdxi2 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*
        dNdxi3 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) + 1)*(xi
      case 78
        dNdxi1 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*
        dNdxi2 = -(2*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(x
        dNdxi3 = (2*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi
      case 79
        dNdxi1 = (2*xi(3)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
        dNdxi2 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*
        dNdxi3 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi
      case 80
        dNdxi1 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*
        dNdxi2 = (2*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
        dNdxi3 = -(2*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1)*(x
      case 81
        dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1
      case 82
        dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(3) - 1
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(2) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1
      case 83
        dNdxi1 = (8*xi(2)*xi(3)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(1) + 1)*(xi(3) - 1
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(1) + 1)*(xi(2) - 1
      case 84
        dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) + 1)*(xi(3) - 1
        dNdxi2 = (8*xi(1)*xi(3)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1
      case 85
        dNdxi1 = -(8*xi(2)*xi(3)*(4*xi(1) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(3) - 1
        dNdxi3 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(2) - 1
      case 86
        dNdxi1 = -(8*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1
        dNdxi2 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1
        dNdxi3 = (8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)
      case 87
        dNdxi1 = (2*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
        dNdxi2 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*
        dNdxi3 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) - 1)*(xi(1) - 1)*(xi
      case 88
        dNdxi1 = (2*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
        dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(4*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
        dNdxi3 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*
      case 89
```

```
        dNdxi1 = -(2*xi(2)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(x
        dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) + 1)*(xi
        dNdxi3 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(8*xi(3)^2 - 5)*
      case 90
        dNdxi1 = (2*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) + 1)*(xi
        dNdxi2 = -(2*xi(1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(x
        dNdxi3 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*
      case 91
        dNdxi1 = (2*xi(2)*(4*xi(1) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
        dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi
        dNdxi3 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*
      case 92
        dNdxi1 = (2*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi
        dNdxi2 = -(4*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*
        dNdxi3 = -(2*xi(1)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(4*xi(3) + 1)*(xi(1) - 1)*(x
      case 93
        dNdxi1 = (xi(1)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2
        dNdxi2 = (xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2
        dNdxi3 = -((4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 16*xi(3)^3 + 12*xi(
      case 94
        dNdxi1 = (xi(1)*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2
        dNdxi2 = -((4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 16*xi(2)^3 + 12*xi(
        dNdxi3 = (xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2
      case 95
        dNdxi1 = -((4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 16*xi(1)^3 - 12*xi(
        dNdxi2 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2
        dNdxi3 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2
      case 96
        dNdxi1 = (xi(1)*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2
        dNdxi2 = -((4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 16*xi(2)^3 - 12*xi(
        dNdxi3 = (xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2
      case 97
        dNdxi1 = -((4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 16*xi(1)^3 + 12*xi(
        dNdxi2 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2
        dNdxi3 = (xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2
      case 98
        dNdxi1 = (xi(1)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2
        dNdxi2 = (xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2
        dNdxi3 = -((4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 16*xi(3)^3 - 12*xi(
      case 99
        dNdxi1 = (64*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*
        dNdxi2 = (64*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*
        dNdxi3 = (64*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*
      case 100
        dNdxi1 = (32*xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(8*xi(1)^2 - 5)*(xi(2) - 1)*(x
        dNdxi2 = -(16*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
        dNdxi3 = -(16*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
      case 101
        dNdxi1 = (64*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*
        dNdxi2 = (64*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*
        dNdxi3 = (64*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*
      case 102
```

```
      dNdxi1 = -(16*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi
      dNdxi2 = (32*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(8*xi(2)^2 - 5)*(xi(1) - 1)*(x
      dNdxi3 = -(16*xi(1)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
    case 103
      dNdxi1 = -(8*xi(1)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(8*xi(1)^2 - 5)*(xi(2)
      dNdxi2 = -(8*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(8*xi(2)^2 - 5)*(xi(1)
      dNdxi3 = (4*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 8*xi(3)^3 + 3*xi(3
    case 104
      dNdxi1 = -(16*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi
      dNdxi2 = (32*xi(1)*xi(2)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(8*xi(2)^2 - 5)*(xi(1) - 1)*(x
      dNdxi3 = -(16*xi(1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
    case 105
      dNdxi1 = (64*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*
      dNdxi2 = (64*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*
      dNdxi3 = (64*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*
    case 106
      dNdxi1 = (32*xi(1)*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(8*xi(1)^2 - 5)*(xi(2) - 1)*(x
      dNdxi2 = -(16*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
      dNdxi3 = -(16*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
    case 107
      dNdxi1 = (64*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) - 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*
      dNdxi2 = (64*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(3) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*
      dNdxi3 = (64*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*
    case 108
      dNdxi1 = -(16*xi(2)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi
      dNdxi2 = -(16*xi(1)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
      dNdxi3 = (32*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(2) - 1)*(8*xi(3)^2 - 5)*(xi(1) - 1)*(x
    case 109
      dNdxi1 = -(8*xi(1)*xi(2)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*(xi(2)
      dNdxi2 = (4*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 8*xi(2)^3 + 3*xi(2
      dNdxi3 = -(8*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(8*xi(3)^2 - 5)*(xi(1)
    case 110
      dNdxi1 = -(16*xi(2)*(2*xi(2) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi
      dNdxi2 = -(16*xi(1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
      dNdxi3 = (32*xi(1)*xi(2)*xi(3)*(2*xi(1) + 1)*(2*xi(2) - 1)*(8*xi(3)^2 - 5)*(xi(1) - 1)*(x
    case 111
      dNdxi1 = (4*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 8*xi(1)^3 + 3*xi(1
      dNdxi2 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*(xi(1)
      dNdxi3 = -(8*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*(xi(1)
    case 112
      dNdxi1 = 2*xi(1)*(8*xi(1)^2 - 5)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1);
      dNdxi2 = 2*xi(2)*(8*xi(2)^2 - 5)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1);
      dNdxi3 = 2*xi(3)*(8*xi(3)^2 - 5)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1);
    case 113
      dNdxi1 = (4*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 8*xi(1)^3 - 3*xi(1
      dNdxi2 = -(8*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*(xi(1)
      dNdxi3 = -(8*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*(xi(1)
    case 114
      dNdxi1 = -(16*xi(2)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi
      dNdxi2 = -(16*xi(1)*(2*xi(1) - 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
      dNdxi3 = (32*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*(xi(1) - 1)*(x
    case 115
```

```
      dNdxi1 = -(8*xi(1)*xi(2)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*(xi(2)
      dNdxi2 = (4*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(3)^4 - 5*xi(3)^2 + 1)*(- 8*xi(2)^3 - 3*xi(2
      dNdxi3 = -(8*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*(xi(1)
    case 116
      dNdxi1 = -(16*xi(2)*(2*xi(2) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi
      dNdxi2 = -(16*xi(1)*(2*xi(1) + 1)*(2*xi(3) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
      dNdxi3 = (32*xi(1)*xi(2)*xi(3)*(2*xi(1) + 1)*(2*xi(2) + 1)*(8*xi(3)^2 - 5)*(xi(1) - 1)*(x
    case 117
      dNdxi1 = (64*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*
      dNdxi2 = (64*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*
      dNdxi3 = (64*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*
    case 118
      dNdxi1 = (32*xi(1)*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*(xi(2) - 1)*(x
      dNdxi2 = -(16*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
      dNdxi3 = -(16*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
    case 119
      dNdxi1 = (64*xi(2)*xi(3)*(2*xi(2) - 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*
      dNdxi2 = (64*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*
      dNdxi3 = (64*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*
    case 120
      dNdxi1 = -(16*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi
      dNdxi2 = (32*xi(1)*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*(xi(1) - 1)*(x
      dNdxi3 = -(16*xi(1)*(2*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
    case 121
      dNdxi1 = -(8*xi(1)*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*(xi(2)
      dNdxi2 = -(8*xi(2)*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*(xi(1)
      dNdxi3 = (4*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 8*xi(3)^3 - 3*xi(3
    case 122
      dNdxi1 = -(16*xi(3)*(2*xi(2) - 1)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi
      dNdxi2 = (32*xi(1)*xi(2)*xi(3)*(2*xi(1) + 1)*(2*xi(3) + 1)*(8*xi(2)^2 - 5)*(xi(1) - 1)*(x
      dNdxi3 = -(16*xi(1)*(2*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
    case 123
      dNdxi1 = (64*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*
      dNdxi2 = (64*xi(1)*xi(3)*(2*xi(1) - 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*
      dNdxi3 = (64*xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*
    case 124
      dNdxi1 = (32*xi(1)*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) + 1)*(8*xi(1)^2 - 5)*(xi(2) - 1)*(x
      dNdxi2 = -(16*xi(3)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
      dNdxi3 = -(16*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi
    case 125
      dNdxi1 = (64*xi(2)*xi(3)*(2*xi(2) + 1)*(2*xi(3) + 1)*(xi(2) - 1)*(xi(2) + 1)*(xi(3) - 1)*
      dNdxi2 = (64*xi(1)*xi(3)*(2*xi(1) + 1)*(2*xi(3) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(3) - 1)*
      dNdxi3 = (64*xi(1)*xi(2)*(2*xi(1) + 1)*(2*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*
  end

  if( i_eval==2 )

    vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3;

  elseif( i_eval==3 )

    vBase = aInvJac(:,4)*dNdxi1 + aInvJac(:,5)*dNdxi2 + aInvJac(:,6)*dNdxi3;
```

```
    else

      vBase = aInvJac(:,7)*dNdxi1 + aInvJac(:,8)*dNdxi2 + aInvJac(:,9)*dNdxi3;
    end

  case {22,23,24,32,33,34,42,43,44}   % Evaluation of second order derivatives.
    error('sf_hex_Q4: second order derivative evaluation not supported.')

  otherwise
    vBase = 0;

end
```

11.48 ellib/sf_hex_Q5

SF_HEX_Q5 Triquintic conforming shape function for hexahedrons (Q5).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_HEX_Q5(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming triquintic Q5 shape functions on hexahedrons with values defined in the nodes, edges, faces, and cell center. XI is
[-1..1]^3 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  3             Number of space dimensions
n_vert      scalar:  8             Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag5, sf_hex_Q1

Code listing

```
 nLDof = [8 48 96 64];
 xLDof = [-1  1  1 -1 -1  1 1 -1 -3/5    1 3/5  -1   -1    1    1   -1 -3/5    1 3/5
-1 -1/5    1 1/5  -1   -1    1    1   -1 -1/5    1 1/5  -1 1/5   1 -1/5   -1  -1
1   1  -1 1/5   1 -1/5   -1 3/5   1 -3/5   -1  -1   1   1  -1 3/5   1 -3/5   -1 -3/5 -3/5
1  3/5   -1 -3/5 -1/5 -1/5    1  1/5   -1 -1/5  1/5  1/5    1 -1/5   -1  1/5  3/5
3/5    1 -3/5   -1  3/5  3/5  3/5    1 -3/5   -1  3/5 3/5 3/5   1 -3/5   -1 3/5 3/5 3/5
1 -3/5   -1 3/5 1/5 1/5   1 -1/5   -1 1/5 -1/5 -1/5    1 1/5  -1 -1/5 -3/5 -3/5
1 3/5  -1 -3/5 -3/5 -3/5    1 3/5  -1 -3/5 -3/5 -3/5    1  3/5   -1 -3/5 -1/5 -1/5
1  1/5   -1 -1/5  1/5  1/5    1 -1/5   -1  1/5 1/5 1/5   1 -1/5   -1 1/5 -1/5 -1/5
1 1/5  -1 -1/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5
1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5
3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5 1/5 3/5 -3/5 -1/5 1/5 3/5 -3/5 -1/5
1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5 1/5 3/5 -3/5 -1/5 1/5 3/5;
          -1 -1  1  1 -1 -1 1  1   -1 -3/5   1 3/5   -1   -1    1    1   -1 -3/5
1 3/5   -1 -1/5   1 1/5   -1   -1    1    1   -1 -1/5   1 1/5  -1 1/5    1 -1/5  -1
-1   1   1  -1 1/5    1 -1/5  -1 3/5    1 -3/5  -1  -1   1   1  -1 3/5    1 -3/5 -3/5
-1 -3/5    1  3/5 -3/5 -3/5   -1 -1/5    1  1/5 -3/5 -3/5   -1  1/5    1 -1/5 -3/5 -3/5
-1  3/5    1 -3/5 -3/5 -1/5   -1  3/5    1 -3/5 -1/5 1/5  -1 3/5    1 -3/5 1/5 3/5
-1 3/5    1 -3/5 3/5 3/5  -1 1/5    1 -1/5 3/5  3/5   -1 -1/5   1 1/5  3/5  3/5   -1 -3/5
1 3/5  3/5  1/5   -1 -3/5   1 3/5  1/5 -1/5   -1 -3/5    1  3/5 -1/5 -1/5   -1 -1/5
1  1/5 -1/5 -1/5   -1  1/5    1 -1/5 -1/5 1/5  -1 1/5    1 -1/5 1/5  1/5   -1 -1/5
1 1/5  1/5 -3/5 -3/5 -3/5 -3/5 -1/5 -1/5 -1/5 -1/5  1/5  1/5  1/5  1/5  3/5  3/5  3/5
3/5 -3/5 -3/5 -3/5 -3/5 -1/5 -1/5 -1/5 -1/5  1/5  1/5  1/5  1/5  3/5  3/5  3/5  3/5 -3/5 -3/5 -3/5
1/5  1/5 1/5 1/5  3/5  3/5 3/5 3/5 -3/5 -3/5 -3/5 -3/5 -1/5 -1/5 -1/5 -1/5  1/5  1/5 1/5 1/5
3/5  3/5 3/5 3/5;
          -1 -1 -1 -1  1  1 1  1   -1   -1  -1  -1 -3/5 -3/5 -3/5 -3/5    1    1
1   1   -1   -1  -1  -1 -1/5 -1/5 -1/5 -1/5    1    1   1   1  -1  -1   -1   -1 1/5 1/5 1/5 1/5
1   1    1    1  -1  -1   -1   -1 3/5 3/5 3/5 3/5   1   1    1    1   -1 -3/5 -3/5 -3/5 -3/5
1   -1 -3/5 -3/5 -3/5 -3/5    1   -1 -3/5 -3/5 -3/5 -3/5    1   -1 -3/5 -3/5 -3/5 -3/5
1   -1 -1/5 -1/5 -1/5 -1/5    1  -1 1/5 1/5  1/5  1/5   1  -1 3/5 3/5  3/5  3/5   1
-1 3/5 3/5  3/5  3/5   1   -1  3/5  3/5 3/5 3/5    1   -1  3/5  3/5 3/5 3/5    1
-1  1/5  1/5 1/5 1/5    1   -1 -1/5 -1/5 -1/5 -1/5    1   -1 -1/5 -1/5 -1/5 -1/5
1   -1 -1/5 -1/5 -1/5 -1/5    1  -1 1/5 1/5  1/5  1/5   1   -1  1/5  1/5 1/5 1/5
1 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -1/5 -1/5 -1/5 -
1/5  1/5  1/5  1/5  1/5  1/5  1/5  1/5  1/5  1/5 1/5 1/5  1/5  1/5 1/5 1/5  3/5  3/5
3/5  3/5  3/5  3/5  3/5  3/5  3/5  3/5 3/5 3/5  3/5  3/5 3/5 3/5];
 sfun  = 'sf_hex_Q5';


 switch i_eval

   case 1
```

```
    switch i_dof

      case 1
        vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2
      case 2
        vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2)
      case 3
        vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2
      case 4
        vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2)
      case 5
        vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2)
      case 6
        vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2
      case 7
        vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2)
      case 8
        vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2
      case 9
        vBase = ((625*xi(2)^5 - 625*xi(2)^4 - 250*xi(2)^3 + 250*xi(2)^2 + 9*xi(2) - 9)*(625*xi(3)
      case 10
        vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(3
      case 11
        vBase = ((625*xi(2)^5 + 625*xi(2)^4 - 250*xi(2)^3 - 250*xi(2)^2 + 9*xi(2) + 9)*(625*xi(3)
      case 12
        vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(3
      case 13
        vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2)
      case 14
        vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2
      case 15
        vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2)
      case 16
        vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2
      case 17
        vBase = -((625*xi(2)^5 - 625*xi(2)^4 - 250*xi(2)^3 + 250*xi(2)^2 + 9*xi(2) - 9)*(625*xi(3
      case 18
        vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(3)
      case 19
        vBase = -((625*xi(2)^5 + 625*xi(2)^4 - 250*xi(2)^3 - 250*xi(2)^2 + 9*xi(2) + 9)*(625*xi(3
      case 20
        vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(3)
      case 21
        vBase = -((625*xi(2)^5 - 625*xi(2)^4 - 250*xi(2)^3 + 250*xi(2)^2 + 9*xi(2) - 9)*(625*xi(3
      case 22
        vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(3)
      case 23
        vBase = -((625*xi(2)^5 + 625*xi(2)^4 - 250*xi(2)^3 - 250*xi(2)^2 + 9*xi(2) + 9)*(625*xi(3
      case 24
        vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(3)
      case 25
```

```
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2
        case 26
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2)
        case 27
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2
        case 28
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2)
        case 29
            vBase = ((625*xi(2)^5 - 625*xi(2)^4 - 250*xi(2)^3 + 250*xi(2)^2 + 9*xi(2) - 9)*(625*xi(3)
        case 30
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(3
        case 31
            vBase = ((625*xi(2)^5 + 625*xi(2)^4 - 250*xi(2)^3 - 250*xi(2)^2 + 9*xi(2) + 9)*(625*xi(3)
        case 32
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(3
        case 33
            vBase = ((625*xi(2)^5 - 625*xi(2)^4 - 250*xi(2)^3 + 250*xi(2)^2 + 9*xi(2) - 9)*(625*xi(3)
        case 34
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(3
        case 35
            vBase = ((625*xi(2)^5 + 625*xi(2)^4 - 250*xi(2)^3 - 250*xi(2)^2 + 9*xi(2) + 9)*(625*xi(3)
        case 36
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(3
        case 37
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2)
        case 38
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2
        case 39
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2)
        case 40
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2
        case 41
            vBase = -((625*xi(2)^5 - 625*xi(2)^4 - 250*xi(2)^3 + 250*xi(2)^2 + 9*xi(2) - 9)*(625*xi(3
        case 42
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(3)
        case 43
            vBase = -((625*xi(2)^5 + 625*xi(2)^4 - 250*xi(2)^3 - 250*xi(2)^2 + 9*xi(2) + 9)*(625*xi(3
        case 44
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(3)
        case 45
            vBase = -((625*xi(2)^5 - 625*xi(2)^4 - 250*xi(2)^3 + 250*xi(2)^2 + 9*xi(2) - 9)*(625*xi(3
        case 46
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(3)
        case 47
            vBase = -((625*xi(2)^5 + 625*xi(2)^4 - 250*xi(2)^3 - 250*xi(2)^2 + 9*xi(2) + 9)*(625*xi(3
        case 48
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(3)
        case 49
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2
        case 50
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2)
        case 51
```

```
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(2
          case 52
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(2)
          case 53
            vBase = ((625*xi(2)^5 - 625*xi(2)^4 - 250*xi(2)^3 + 250*xi(2)^2 + 9*xi(2) - 9)*(625*xi(3)
          case 54
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(625*xi(3
          case 55
            vBase = ((625*xi(2)^5 + 625*xi(2)^4 - 250*xi(2)^3 - 250*xi(2)^2 + 9*xi(2) + 9)*(625*xi(3)
          case 56
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(625*xi(3
          case 57
            vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(3)^
          case 58
            vBase = -((125*xi(3)^5 - 75*xi(3)^4 - 130*xi(3)^3 + 78*xi(3)^2 + 5*xi(3) - 3)*(625*xi(2)^
          case 59
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3)
          case 60
            vBase = -((125*xi(3)^5 - 75*xi(3)^4 - 130*xi(3)^3 + 78*xi(3)^2 + 5*xi(3) - 3)*(625*xi(2)^
          case 61
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3)
          case 62
            vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(3)^5
          case 63
            vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(3)^5
          case 64
            vBase = ((125*xi(3)^5 - 75*xi(3)^4 - 130*xi(3)^3 + 78*xi(3)^2 + 5*xi(3) - 3)*(625*xi(2)^5
          case 65
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3
          case 66
            vBase = ((125*xi(3)^5 - 75*xi(3)^4 - 130*xi(3)^3 + 78*xi(3)^2 + 5*xi(3) - 3)*(625*xi(2)^5
          case 67
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3
          case 68
            vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(3)^
          case 69
            vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(3)^
          case 70
            vBase = -((125*xi(3)^5 - 75*xi(3)^4 - 130*xi(3)^3 + 78*xi(3)^2 + 5*xi(3) - 3)*(625*xi(2)^
          case 71
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3)
          case 72
            vBase = -((125*xi(3)^5 - 75*xi(3)^4 - 130*xi(3)^3 + 78*xi(3)^2 + 5*xi(3) - 3)*(625*xi(2)^
          case 73
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3)
          case 74
            vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(3)^5
          case 75
            vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(3)^5
          case 76
            vBase = ((125*xi(3)^5 - 75*xi(3)^4 - 130*xi(3)^3 + 78*xi(3)^2 + 5*xi(3) - 3)*(625*xi(2)^5
          case 77
```

```
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3
          case 78
            vBase = ((125*xi(3)^5 - 75*xi(3)^4 - 130*xi(3)^3 + 78*xi(3)^2 + 5*xi(3) - 3)*(625*xi(2)^5
          case 79
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3
          case 80
            vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(3)^
          case 81
            vBase = -((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(625*xi(3)^
          case 82
            vBase = -((125*xi(3)^5 - 25*xi(3)^4 - 170*xi(3)^3 + 34*xi(3)^2 + 45*xi(3) - 9)*(625*xi(2)
          case 83
            vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(1)^5
          case 84
            vBase = -((125*xi(3)^5 - 25*xi(3)^4 - 170*xi(3)^3 + 34*xi(3)^2 + 45*xi(3) - 9)*(625*xi(2)
          case 85
            vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(1)^5
          case 86
            vBase = ((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(625*xi(3)^5
          case 87
            vBase = ((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(625*xi(3)^5
          case 88
            vBase = ((125*xi(3)^5 + 25*xi(3)^4 - 170*xi(3)^3 - 34*xi(3)^2 + 45*xi(3) + 9)*(625*xi(2)^
          case 89
            vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(1)^
          case 90
            vBase = ((125*xi(3)^5 + 25*xi(3)^4 - 170*xi(3)^3 - 34*xi(3)^2 + 45*xi(3) + 9)*(625*xi(2)^
          case 91
            vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(1)^
          case 92
            vBase = -((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(625*xi(3)^
          case 93
            vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(3)^
          case 94
            vBase = -((125*xi(3)^5 + 75*xi(3)^4 - 130*xi(3)^3 - 78*xi(3)^2 + 5*xi(3) + 3)*(625*xi(2)^
          case 95
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3)
          case 96
            vBase = -((125*xi(3)^5 + 75*xi(3)^4 - 130*xi(3)^3 - 78*xi(3)^2 + 5*xi(3) + 3)*(625*xi(2)^
          case 97
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3)
          case 98
            vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(3)^5
          case 99
            vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(3)^5
          case 100
            vBase = ((125*xi(3)^5 + 75*xi(3)^4 - 130*xi(3)^3 - 78*xi(3)^2 + 5*xi(3) + 3)*(625*xi(2)^5
          case 101
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3
          case 102
            vBase = ((125*xi(3)^5 + 75*xi(3)^4 - 130*xi(3)^3 - 78*xi(3)^2 + 5*xi(3) + 3)*(625*xi(2)^5
          case 103
```

```
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3
          case 104
            vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(3)^
          case 105
            vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(3)^
          case 106
            vBase = -((125*xi(3)^5 + 75*xi(3)^4 - 130*xi(3)^3 - 78*xi(3)^2 + 5*xi(3) + 3)*(625*xi(2)^
          case 107
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3)
          case 108
            vBase = -((125*xi(3)^5 + 75*xi(3)^4 - 130*xi(3)^3 - 78*xi(3)^2 + 5*xi(3) + 3)*(625*xi(2)^
          case 109
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3)
          case 110
            vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(3)^5
          case 111
            vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(3)^5
          case 112
            vBase = ((125*xi(3)^5 + 75*xi(3)^4 - 130*xi(3)^3 - 78*xi(3)^2 + 5*xi(3) + 3)*(625*xi(2)^5
          case 113
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3
          case 114
            vBase = ((125*xi(3)^5 + 75*xi(3)^4 - 130*xi(3)^3 - 78*xi(3)^2 + 5*xi(3) + 3)*(625*xi(2)^5
          case 115
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3
          case 116
            vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(3)^
          case 117
            vBase = -((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(625*xi(3)^
          case 118
            vBase = -((125*xi(3)^5 + 25*xi(3)^4 - 170*xi(3)^3 - 34*xi(3)^2 + 45*xi(3) + 9)*(625*xi(2)
          case 119
            vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(1)^5
          case 120
            vBase = -((125*xi(3)^5 + 25*xi(3)^4 - 170*xi(3)^3 - 34*xi(3)^2 + 45*xi(3) + 9)*(625*xi(2)
          case 121
            vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(1)^5
          case 122
            vBase = ((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(625*xi(3)^5
          case 123
            vBase = ((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(625*xi(3)^5
          case 124
            vBase = ((125*xi(3)^5 - 25*xi(3)^4 - 170*xi(3)^3 + 34*xi(3)^2 + 45*xi(3) - 9)*(625*xi(2)^
          case 125
            vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(625*xi(1)^
          case 126
            vBase = ((125*xi(3)^5 - 25*xi(3)^4 - 170*xi(3)^3 + 34*xi(3)^2 + 45*xi(3) - 9)*(625*xi(2)^
          case 127
            vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(625*xi(1)^
          case 128
            vBase = -((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(625*xi(3)^
          case 129
```

```
            vBase = -((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(625*xi(3)
          case 130
            vBase = -((125*xi(3)^5 - 25*xi(3)^4 - 170*xi(3)^3 + 34*xi(3)^2 + 45*xi(3) - 9)*(625*xi(2)
          case 131
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3)
          case 132
            vBase = -((125*xi(3)^5 - 25*xi(3)^4 - 170*xi(3)^3 + 34*xi(3)^2 + 45*xi(3) - 9)*(625*xi(2)
          case 133
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3)
          case 134
            vBase = ((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(625*xi(3)^
          case 135
            vBase = ((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(625*xi(3)^
          case 136
            vBase = ((125*xi(3)^5 - 25*xi(3)^4 - 170*xi(3)^3 + 34*xi(3)^2 + 45*xi(3) - 9)*(625*xi(2)^
          case 137
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3
          case 138
            vBase = ((125*xi(3)^5 - 25*xi(3)^4 - 170*xi(3)^3 + 34*xi(3)^2 + 45*xi(3) - 9)*(625*xi(2)^
          case 139
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3
          case 140
            vBase = -((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(625*xi(3)
          case 141
            vBase = -((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(625*xi(3)
          case 142
            vBase = -((125*xi(3)^5 + 25*xi(3)^4 - 170*xi(3)^3 - 34*xi(3)^2 + 45*xi(3) + 9)*(625*xi(2)
          case 143
            vBase = ((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3)
          case 144
            vBase = -((125*xi(3)^5 + 25*xi(3)^4 - 170*xi(3)^3 - 34*xi(3)^2 + 45*xi(3) + 9)*(625*xi(2)
          case 145
            vBase = ((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3)
          case 146
            vBase = ((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(625*xi(3)^
          case 147
            vBase = ((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(625*xi(3)^
          case 148
            vBase = ((125*xi(3)^5 + 25*xi(3)^4 - 170*xi(3)^3 - 34*xi(3)^2 + 45*xi(3) + 9)*(625*xi(2)^
          case 149
            vBase = -((625*xi(1)^5 + 625*xi(1)^4 - 250*xi(1)^3 - 250*xi(1)^2 + 9*xi(1) + 9)*(125*xi(3
          case 150
            vBase = ((125*xi(3)^5 + 25*xi(3)^4 - 170*xi(3)^3 - 34*xi(3)^2 + 45*xi(3) + 9)*(625*xi(2)^
          case 151
            vBase = -((625*xi(1)^5 - 625*xi(1)^4 - 250*xi(1)^3 + 250*xi(1)^2 + 9*xi(1) - 9)*(125*xi(3
          case 152
            vBase = -((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(625*xi(3)
          case 153
            vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^5
          case 154
            vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^
          case 155
```

```
        vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^5
    case 156
        vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^
    case 157
        vBase = -((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(125*xi(3)^
    case 158
        vBase = ((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(125*xi(3)^
    case 159
        vBase = -((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(125*xi(3)
    case 160
        vBase = ((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(125*xi(3)^5
    case 161
        vBase = ((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(125*xi(3)^5
    case 162
        vBase = -((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(125*xi(3)
    case 163
        vBase = ((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(125*xi(3)^
    case 164
        vBase = -((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(125*xi(3)^
    case 165
        vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^
    case 166
        vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^5
    case 167
        vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^
    case 168
        vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^5
    case 169
        vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^
    case 170
        vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^5
    case 171
        vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^
    case 172
        vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^5
    case 173
        vBase = ((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(125*xi(3)^5
    case 174
        vBase = -((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(125*xi(3)
    case 175
        vBase = ((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(125*xi(3)^
    case 176
        vBase = -((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(125*xi(3)^
    case 177
        vBase = -((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(125*xi(3)^
    case 178
        vBase = ((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(125*xi(3)^
    case 179
        vBase = -((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(125*xi(3)
    case 180
        vBase = ((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(125*xi(3)^5
    case 181
```

```
          vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^5
        case 182
          vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^
        case 183
          vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^5
        case 184
          vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^
        case 185
          vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^5
        case 186
          vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^
        case 187
          vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^5
        case 188
          vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^
        case 189
          vBase = -((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(125*xi(3)^
        case 190
          vBase = ((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(125*xi(3)^
        case 191
          vBase = -((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(125*xi(3)
        case 192
          vBase = ((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(125*xi(3)^5
        case 193
          vBase = ((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(125*xi(3)^5
        case 194
          vBase = -((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(125*xi(3)
        case 195
          vBase = ((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(125*xi(3)^
        case 196
          vBase = -((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(125*xi(3)^
        case 197
          vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^
        case 198
          vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^5
        case 199
          vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^
        case 200
          vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^5
        case 201
          vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^
        case 202
          vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^5
        case 203
          vBase = -((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^
        case 204
          vBase = ((125*xi(2)^5 - 75*xi(2)^4 - 130*xi(2)^3 + 78*xi(2)^2 + 5*xi(2) - 3)*(125*xi(3)^5
        case 205
          vBase = ((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(125*xi(3)^5
        case 206
          vBase = -((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(125*xi(3)
        case 207
```

```
          vBase = ((125*xi(2)^5 - 25*xi(2)^4 - 170*xi(2)^3 + 34*xi(2)^2 + 45*xi(2) - 9)*(125*xi(3)^
        case 208
          vBase = -((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(125*xi(3)^
        case 209
          vBase = -((125*xi(1)^5 - 75*xi(1)^4 - 130*xi(1)^3 + 78*xi(1)^2 + 5*xi(1) - 3)*(125*xi(3)^
        case 210
          vBase = ((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(125*xi(3)^
        case 211
          vBase = -((125*xi(2)^5 + 25*xi(2)^4 - 170*xi(2)^3 - 34*xi(2)^2 + 45*xi(2) + 9)*(125*xi(3)
        case 212
          vBase = ((125*xi(1)^5 + 75*xi(1)^4 - 130*xi(1)^3 - 78*xi(1)^2 + 5*xi(1) + 3)*(125*xi(3)^5
        case 213
          vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^5
        case 214
          vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^
        case 215
          vBase = ((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^5
        case 216
          vBase = -((125*xi(2)^5 + 75*xi(2)^4 - 130*xi(2)^3 - 78*xi(2)^2 + 5*xi(2) + 3)*(125*xi(3)^
      end

    case {2,3,4}

      switch i_dof
        case 1
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 2
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 3
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 4
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 5
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 6
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 7
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
```

```
        case 8
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 9
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(
        case 10
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 11
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(
        case 12
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 13
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 14
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 15
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 16
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 17
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 18
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(
        case 19
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 20
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(
```

```
        case 21
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 22
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(
        case 23
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -(25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 24
          dNdxi1 = (25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(
        case 25
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 26
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 27
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 28
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 29
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(
        case 30
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 31
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = (25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(
        case 32
          dNdxi1 = -(25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi
        case 33
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = (25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(
```

```
        case 34
          dNdxi1 = -(25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi
        case 35
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(
        case 36
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 37
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 38
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 39
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 40
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 41
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -(25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 42
          dNdxi1 = (25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(
        case 43
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 44
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(
        case 45
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 46
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(
```

```
        case 47
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 48
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(
        case 49
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 50
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 51
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 52
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 53
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(
        case 54
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 55
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(
          dNdxi3 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(
        case 56
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 57
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 58
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 59
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
```

```
        case 60
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 61
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 62
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 63
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 64
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 65
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*x
        case 66
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 67
          dNdxi1 = -(625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*x
        case 68
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 69
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 70
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 71
          dNdxi1 = (625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi
        case 72
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
```

```
        case 73
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 74
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 75
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 76
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 77
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*x
        case 78
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 79
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*x
        case 80
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 81
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*x
        case 82
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 83
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 84
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 85
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
```

```
        case 86
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi
        case 87
          dNdxi1 = (625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi
        case 88
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 89
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*x
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 90
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 91
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*x
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 92
          dNdxi1 = -(625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*x
        case 93
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 94
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 95
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 96
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 97
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 98
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
```

```
        case 99
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 100
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 101
          dNdxi1 = -(625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*x
        case 102
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 103
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*x
        case 104
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 105
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 106
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 107
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 108
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 109
          dNdxi1 = (625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi
        case 110
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 111
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
```

```
        case 112
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 113
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*x
        case 114
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 115
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*x
        case 116
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 117
          dNdxi1 = -(625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*x
        case 118
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 119
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 120
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 121
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 122
          dNdxi1 = (625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi
        case 123
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi
        case 124
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
```

```
        case 125
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*x
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*x
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 126
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 127
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*x
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*x
          dNdxi3 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2)
        case 128
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*x
        case 129
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*x
        case 130
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 131
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 132
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*x
          dNdxi3 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 133
          dNdxi1 = (625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi
        case 134
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi
        case 135
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi
        case 136
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi
          dNdxi3 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 137
          dNdxi1 = -(625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*x
```

```
        case 138
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 139
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*x
        case 140
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*x
        case 141
          dNdxi1 = -((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*x
        case 142
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*x
          dNdxi3 = -(625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 143
          dNdxi1 = (625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi
        case 144
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*x
        case 145
          dNdxi1 = (625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi
        case 146
          dNdxi1 = ((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi
        case 147
          dNdxi1 = ((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi
        case 148
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi
          dNdxi3 = (625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
        case 149
          dNdxi1 = -(625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*x
        case 150
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi
          dNdxi3 = (625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi
```

```
        case 151
          dNdxi1 = -(625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*x
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*x
        case 152
          dNdxi1 = -((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*x
          dNdxi3 = -(625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*x
        case 153
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 154
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 155
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 156
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 157
          dNdxi1 = -(15625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5
        case 158
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*
        case 159
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5
        case 160
          dNdxi1 = (15625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*
        case 161
          dNdxi1 = (15625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*
        case 162
          dNdxi1 = -((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5
        case 163
          dNdxi1 = ((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*
```

```
        case 164
          dNdxi1 = -(15625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5
        case 165
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 166
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 167
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 168
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 169
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 170
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 171
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 172
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 173
          dNdxi1 = (15625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*
        case 174
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5
        case 175
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*
        case 176
          dNdxi1 = -(15625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5
```

```
        case 177
          dNdxi1 = -(15625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5
        case 178
          dNdxi1 = ((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*
        case 179
          dNdxi1 = -((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5
        case 180
          dNdxi1 = (15625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*
        case 181
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 182
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 183
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 184
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 185
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 186
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 187
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 188
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 189
          dNdxi1 = -(15625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5
```

```
        case 190
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*
        case 191
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5
        case 192
          dNdxi1 = (15625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*
        case 193
          dNdxi1 = (15625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*
        case 194
          dNdxi1 = -((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5
        case 195
          dNdxi1 = ((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*
        case 196
          dNdxi1 = -(15625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5
        case 197
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 198
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 199
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5
          dNdxi3 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 200
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) + 1)*(5*xi(3) - 3)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 201
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 202
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
```

```
        case 203
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 204
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 205
          dNdxi1 = (15625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*
        case 206
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5
        case 207
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*
        case 208
          dNdxi1 = -(15625*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) - 3)*(5
        case 209
          dNdxi1 = -(15625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5
        case 210
          dNdxi1 = ((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*
        case 211
          dNdxi1 = -((5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5
        case 212
          dNdxi1 = (15625*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*
        case 213
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
        case 214
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
        case 215
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*
          dNdxi3 = (15625*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*
```

```
        case 216
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5*xi(3)
          dNdxi2 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(3) - 1)*(5*xi(3) + 1)*(5
          dNdxi3 = -(15625*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(5*xi(2) - 1)*(5*xi(2) + 1)*(5
      end

      if( i_eval==2 )

        vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3;

      elseif( i_eval==3 )

        vBase = aInvJac(:,4)*dNdxi1 + aInvJac(:,5)*dNdxi2 + aInvJac(:,6)*dNdxi3;

      else

        vBase = aInvJac(:,7)*dNdxi1 + aInvJac(:,8)*dNdxi2 + aInvJac(:,9)*dNdxi3;

      end

    case {22,23,24,32,33,34,42,43,44}   % Evaluation of second order derivatives.
      error('sf_hex_Q5: second order derivative evaluation not supported.')

    otherwise
      vBase = 0;

  end
```

11.49 ellib/sf_line_H3

SF_LINE_H3 Third order 1D C1 Hermite shape functions for lines.

[VBASE, NLDOF, XLDOF, SFUN] =

SF_LINE_H3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates C1 Hermite shape functions on 1D line elements with value and first derivatives defined in the nodes. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1              Number of space dimensions
n_vert      scalar: 2              Number of vertices per cell
i_dof       scalar: 1-4            Local basis function to evaluate
xi          array  [2,1]           Local coordinates of evaluation point
aInvJac     [n,3]                  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
```

```
Output      Value/[Size]          Description
-------------------------------------------------------------------------------
vBase       [n]                   Evaluated function values
nLDof       [2,4]                 Number of local degrees of freedom on
                                  vertices, edges, faces, cell interiors,
                                  and vertices without boundary conditions
xLDof       [2,n_ldof]            Local coordinates of local dofs
sfun        string                Function name of called shape function
```

See also

sf_line_P3

Code listing

```
nLDof = [2 0 0 0;
         2 0 0 0];
xLDof = [1 0 1 0;
         0 1 0 1];
sfun  = 'sf_line_H3';


switch i_eval    % Evaluation type flag.

  case 1   % Evaluation of function values.

    switch i_dof   % Basis function to evaluate.

      case 1
        vBase = 3*xi(1)^2 - 2*xi(1)^3;
      case 2
        vBase = 3*xi(2)^2 - 2*xi(2)^3;
      case 3
        vBase = ( xi(1)^2 - xi(1)^3 ) .* aInvJac(:,3);
      case 4
        vBase = ( xi(2)^3 - xi(2)^2 ) .* aInvJac(:,3);
    end


  case 2   % Evaluation of first derivatives.

    switch i_dof   % Basis function derivative to evaluate.
      case 1
        dNdxi1 = -6*xi(1)*(xi(1) - 1);
```

```
          dNdxi2 =  0;
        case 2
          dNdxi1 =  0;
          dNdxi2 = -6*xi(2)*(xi(2) - 1);
        case 3
          dNdxi1 = -xi(1)*(3*xi(1) - 2) .* aInvJac(:,3);
          dNdxi2 =  0;
        case 4
          dNdxi1 =  0;
          dNdxi2 =  xi(2)*(3*xi(2) - 2) .* aInvJac(:,3);
      end

      vBase = aInvJac(:,1) .* dNdxi1 + aInvJac(:,2) .* dNdxi2;


    case 22   % Evaluation of second derivatives.

      switch i_dof   % Basis function derivative to evaluate.
        case 1
          dNdxi1 =  6*(2*xi(1)-1) ./ aInvJac(:,3);
          dNdxi2 =  0;
        case 2
          dNdxi1 =  0;
          dNdxi2 = -6*(2*xi(2)-1) ./ aInvJac(:,3);
        case 3
          dNdxi1 =  6*xi(1) - 2;
          dNdxi2 =  0;
        case 4
          dNdxi1 =  0;
          dNdxi2 =  6*xi(2) - 2;
      end

      vBase = aInvJac(:,1) .* dNdxi1 + aInvJac(:,2) .* dNdxi2;


    otherwise
      vBase = 0;

  end
```

11.50 ellib/sf_line_P3

SF_LINE_P3 1D Third order Lagrange shape functions for lines (P3).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_LINE_P3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming third order P3 Lagrange shape functions on 1D line elements with values defined in the nodes and center. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1              Number of space dimensions
n_vert      scalar: 2              Number of vertices per cell
i_dof       scalar: 1-4            Local basis function to evaluate
xi          array [2,1]            Local coordinates of evaluation point
aInvJac     [n,3]                  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [2,n_ldof]             Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag3, sf_line_H3

Code listing

```
nLDof = [2 0 0 2];
xLDof = [1 0 2/3 1/3;
         0 1 1/3 2/3];
sfun  = 'sf_line_P3';


switch i_eval    % Evaluation type flag.

  case 1   % Evaluation of function values.
    xi = xi(1);

    switch i_dof   % Basis function to evaluate.

      case 1
        vBase = xi*(2 - 3*xi)*(1 - 3*xi)/2;
      case 2
        vBase = (1 - xi)*(2 - 3*xi)*(1 - 3*xi)/2;
      case 3
        vBase = 9*xi*(1-xi)*(3*xi - 1)/2;
```

```
        case 4
          vBase = 9*xi*(1-xi)*(2 - 3*xi)/2;
      end


    case 2   % Evaluation of first derivative.
      xi = xi(1);

      switch i_dof   % Basis function derivative to evaluate.
        case 1
          dNdxi = (27*xi^2)/2 - 9*xi + 1;
        case 2
          dNdxi = 18*xi - (27*xi^2)/2 - 11/2;
        case 3
          dNdxi = 36*xi - (81*xi^2)/2 - 9/2;
        case 4
          dNdxi = (81*xi^2)/2 - 45*xi + 9;
      end

      vBase = aInvJac(:,1)*dNdxi;

    case 22   % Evaluation of second derivatives.
      xi = xi(1);

      switch i_dof   % Basis function derivative to evaluate.
        case 1
          dNdxi = 27*xi - 9;
        case 2
          dNdxi = 18 - 27*xi;
        case 3
          dNdxi = 36 - 81*xi;
        case 4
          dNdxi = 81*xi - 45;
      end

      vBase = -aInvJac(:,1)./aInvJac(:,3)*dNdxi;

    otherwise
      vBase = 0;

  end
```

11.51 ellib/sf_line_P4

SF_LINE_P4 1D Fourth order Lagrange shape functions for lines (P4).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_LINE_P4(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming

fourth order P4 Lagrange shape functions on 1D line elements with values defined in the nodes and center. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1              Number of space dimensions
n_vert      scalar: 2              Number of vertices per cell
i_dof       scalar: 1-5            Local basis function to evaluate
xi          array [2,1]            Local coordinates of evaluation point
aInvJac     [n,3]                  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [2,n_ldof]             Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_line_P1

Code listing

```
nLDof = [2 0 0 3];
xLDof = [1 0 3/4 1/2 1/4;
         0 1 1/4 1/2 3/4];
sfun  = 'sf_line_P4';


switch i_eval    % Evaluation type flag.

  case 1   % Evaluation of function values.

    switch i_dof   % Basis function to evaluate.

      case 1
        vBase = (32*xi(1)^4)/3 - 16*xi(1)^3 + (22*xi(1)^2)/3 - xi(1);
      case 2
        vBase = (32*xi(1)^4)/3 - (80*xi(1)^3)/3 + (70*xi(1)^2)/3 - (25*xi(1))/3 + 1;
```

```
        case 3
          vBase = - (128*xi(1)^4)/3 + (224*xi(1)^3)/3 - (112*xi(1)^2)/3 + (16*xi(1))/3;
        case 4
          vBase = 64*xi(1)^4 - 128*xi(1)^3 + 76*xi(1)^2 - 12*xi(1);
        case 5
          vBase = - (128*xi(1)^4)/3 + 96*xi(1)^3 - (208*xi(1)^2)/3 + 16*xi(1);
      end

    case 2   % Evaluation of first derivative.

      switch i_dof   % Basis function derivative to evaluate.

        case 1
          dNdxi1 = ((8*xi(1) - 3)*(16*xi(1)^2 - 12*xi(1) + 1))/3;
        case 2
          dNdxi1 = ((8*xi(1) - 5)*(16*xi(1)^2 - 20*xi(1) + 5))/3;
        case 3
          dNdxi1 = - (512*xi(1)^3)/3 + 224*xi(1)^2 - (224*xi(1))/3 + 16/3;
        case 4
          dNdxi1 = 4*(2*xi(1) - 1)*(32*xi(1)^2 - 32*xi(1) + 3);
        case 5
          dNdxi1 = - (512*xi(1)^3)/3 + 288*xi(1)^2 - (416*xi(1))/3 + 16;
      end

      vBase = aInvJac(:,1) * dNdxi1;

    case 22   % Evaluation of second derivatives.

      switch i_dof   % Basis function derivative to evaluate.

        case 1
          dNdxi1 = 128*xi(1)^2 - 96*xi(1) + 44/3;
        case 2
          dNdxi1 = 128*xi(1)^2 - 160*xi(1) + 140/3;
        case 3
          dNdxi1 = - 512*xi(1)^2 + 448*xi(1) - 224/3;
        case 4
          dNdxi1 = 768*xi(1)^2 - 768*xi(1) + 152;
        case 5
          dNdxi1 = - 512*xi(1)^2 + 576*xi(1) - 416/3;
      end

      vBase = -aInvJac(:,1) ./ aInvJac(:,3) * dNdxi1;

    otherwise
      vBase = 0;

  end
```

11.52 ellib/sf_line_P5

SF_LINE_P5 1D Fifth order Lagrange shape functions for lines (P5).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_LINE_P5(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming fifth order P5 Lagrange shape functions on 1D line elements with values defined in the nodes and center. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1              Number of space dimensions
n_vert      scalar: 2              Number of vertices per cell
i_dof       scalar: 1-6            Local basis function to evaluate
xi          array [2,1]            Local coordinates of evaluation point
aInvJac     [n,3]                  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [2,n_ldof]             Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_line_P1

Code listing

```
nLDof = [2 0 0 4];
xLDof = [1 0 4/5 3/5 2/5 1/5;
         0 1 1/5 2/5 3/5 4/5];
sfun  = 'sf_line_P5';


switch i_eval    % Evaluation type flag.
```

```
    case 1   % Evaluation of function values.

      switch i_dof   % Basis function to evaluate.

        case 1
          vBase = (625*xi(1)^5)/24 - (625*xi(1)^4)/12 + (875*xi(1)^3)/24 - (125*xi(1)^2)/12 + xi(1)
        case 2
          vBase = - (625*xi(1)^5)/24 + (625*xi(1)^4)/8 - (2125*xi(1)^3)/24 + (375*xi(1)^2)/8 - (137
        case 3
          vBase = - (3125*xi(1)^5)/24 + (6875*xi(1)^4)/24 - (5125*xi(1)^3)/24 + (1525*xi(1)^2)/24 -
        case 4
          vBase = (3125*xi(1)^5)/12 - 625*xi(1)^4 + (6125*xi(1)^3)/12 - (325*xi(1)^2)/2 + (50*xi(1)
        case 5
          vBase = - (3125*xi(1)^5)/12 + (8125*xi(1)^4)/12 - (7375*xi(1)^3)/12 + (2675*xi(1)^2)/12 -
        case 6
          vBase = (3125*xi(1)^5)/24 - (4375*xi(1)^4)/12 + (8875*xi(1)^3)/24 - (1925*xi(1)^2)/12 + 2
      end

    case 2   % Evaluation of first derivative.

      switch i_dof   % Basis function derivative to evaluate.

        case 1
          dNdxi1 = (3125*xi(1)^4)/24 - (625*xi(1)^3)/3 + (875*xi(1)^2)/8 - (125*xi(1))/6 + 1;
        case 2
          dNdxi1 = - (3125*xi(1)^4)/24 + (625*xi(1)^3)/2 - (2125*xi(1)^2)/8 + (375*xi(1))/4 - 137/1
        case 3
          dNdxi1 = - (15625*xi(1)^4)/24 + (6875*xi(1)^3)/6 - (5125*xi(1)^2)/8 + (1525*xi(1))/12 - 2
        case 4
          dNdxi1 = (15625*xi(1)^4)/12 - 2500*xi(1)^3 + (6125*xi(1)^2)/4 - 325*xi(1) + 50/3;
        case 5
          dNdxi1 = - (15625*xi(1)^4)/12 + (8125*xi(1)^3)/3 - (7375*xi(1)^2)/4 + (2675*xi(1))/6 - 25
        case 6
          dNdxi1 = (15625*xi(1)^4)/24 - (4375*xi(1)^3)/3 + (8875*xi(1)^2)/8 - (1925*xi(1))/6 + 25;
      end

      vBase = aInvJac(:,1) * dNdxi1;

    case 22   % Evaluation of second derivatives.

      switch i_dof   % Basis function derivative to evaluate.

        case 1
          dNdxi1 = (125*(5*xi(1) - 2)*(10*xi(1)^2 - 8*xi(1) + 1))/12;
        case 2
          dNdxi1 = -(125*(5*xi(1) - 3)*(10*xi(1)^2 - 12*xi(1) + 3))/12;
        case 3
          dNdxi1 = - (15625*xi(1)^3)/6 + (6875*xi(1)^2)/2 - (5125*xi(1))/4 + 1525/12;
        case 4
          dNdxi1 = (15625*xi(1)^3)/3 - 7500*xi(1)^2 + (6125*xi(1))/2 - 325;
        case 5
          dNdxi1 = - (15625*xi(1)^3)/3 + 8125*xi(1)^2 - (7375*xi(1))/2 + 2675/6;
```

```
        case 6
          dNdxi1 = (15625*xi(1)^3)/6 - 4375*xi(1)^2 + (8875*xi(1))/4 - 1925/6;
      end

      vBase = -aInvJac(:,1) ./ aInvJac(:,3) * dNdxi1;

    otherwise
      vBase = 0;

  end
```

11.53 ellib/sf_quad_H3

SF_QUAD_H3 Third order 2D C1 Hermite shape function for quadrilaterals.

[VBASE, NLDOF, XLDOF, SFUN] =

SF_QUAD_H3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates C1 Hermite shape functions on quadrilaterals with values defined in the nodes, cell center. XI are [-1..1]^2 reference coordinates. (Limited to rectangular grid cells.)

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  2             Number of space dimensions
n_vert      scalar:  4             Number of vertices per cell
i_dof       scalar: 1-16           Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [3,4]                  Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_quad_Q3

Code listing

```
nLDof = [4 0 0 4;
         4 0 0 0;
         4 0 0 0];
xLDof = [repmat([-1  1 1 -1;
                 -1 -1 1  1],1,3) [-1 1 1 -1;-1 -1 1 1]/3];
sfun  = 'sf_quad_H3';


if( nargin>=6 && ~isempty(aInvJac) )

  if( any(any(abs(aInvJac(:,2:3))>eps*1e2)) )

    aJTmp = tfjacquad( -2 );
    if( i_dof==5 || i_dof==9 )
      a11 =  aJTmp(:,7)/2;
      a12 = -aJTmp(:,6)/2;
      a21 = -aJTmp(:,3)/2;
      a22 =  aJTmp(:,2)/2;
    end
    if( i_dof==6 || i_dof==10 )
      a11 =  aJTmp(:,7)/2;
      a12 = -aJTmp(:,5)/2;
      a21 = -aJTmp(:,3)/2;
      a22 =  aJTmp(:,1)/2;
    end
    if( i_dof==7 || i_dof==11 )
      a11 =  aJTmp(:,8)/2;
      a12 = -aJTmp(:,5)/2;
      a21 = -aJTmp(:,4)/2;
      a22 =  aJTmp(:,1)/2;
    end
    if( i_dof==8 || i_dof==12 )
      a11 =  aJTmp(:,8)/2;
      a12 = -aJTmp(:,6)/2;
      a21 = -aJTmp(:,4)/2;
      a22 =  aJTmp(:,2)/2;
    end

  else

    a11 = 1./aInvJac(:,1);
    a12 = 0;
```

```
    a21 = 0;
    a22 = 1./aInvJac(:,4);

  end

end

switch i_eval

  case 1

    switch i_dof
      case 1
        vBase = ((xi(1) - 1)*(xi(2) - 1)*(- 153*xi(1)^2*xi(2)^2 - 36*xi(1)^2*xi(2) + 85*xi(1)^2 -
      case 2
        vBase = ((xi(1) + 1)*(xi(2) - 1)*(153*xi(1)^2*xi(2)^2 + 36*xi(1)^2*xi(2) - 85*xi(1)^2 - 3
      case 3
        vBase = -((xi(1) + 1)*(xi(2) + 1)*(153*xi(1)^2*xi(2)^2 - 36*xi(1)^2*xi(2) - 85*xi(1)^2 -
      case 4
        vBase = -((xi(1) - 1)*(xi(2) + 1)*(- 153*xi(1)^2*xi(2)^2 + 36*xi(1)^2*xi(2) + 85*xi(1)^2
      case 5
        vBase = - (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(1) + 1)*(xi(2) - 1))/64 - (a
      case 6
        vBase = (a12*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) + 1)*(xi(2) - 1)^2*(xi(2) + 1))/64 - (a11
      case 7
        vBase = (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)^2*(xi(2) + 1))/64 + (a12
      case 8
        vBase = (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(1) + 1)*(xi(2) + 1))/64 - (a12
      case 9
        vBase = - (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(1) + 1)*(xi(2) - 1))/64 - (a
      case 10
        vBase = (a22*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) + 1)*(xi(2) - 1)^2*(xi(2) + 1))/64 - (a21
      case 11
        vBase = (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)^2*(xi(2) + 1))/64 + (a22
      case 12
        vBase = (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(1) + 1)*(xi(2) + 1))/64 - (a22
      case 13
        vBase = ((3*xi(2) - 1)*(243*xi(1) - 81)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1))/
      case 14
        vBase = -((3*xi(2) - 1)*(243*xi(1) + 81)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1))
      case 15
        vBase = ((3*xi(2) + 1)*(243*xi(1) + 81)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1))/
      case 16
        vBase = -((3*xi(2) + 1)*(243*xi(1) - 81)*(xi(1) - 1)*(xi(1) + 1)*(xi(2) - 1)*(xi(2) + 1))
    end

  case {2,3}

    switch i_dof
      case 1
        dNdxi1 = ((xi(2) - 1)*(- 459*xi(1)^2*xi(2)^2 - 108*xi(1)^2*xi(2) + 255*xi(1)^2 + 234*xi(1
        dNdxi2 = ((xi(1) - 1)*(- 459*xi(1)^2*xi(2)^2 + 234*xi(1)^2*xi(2) + 121*xi(1)^2 - 108*xi(1
```

```
      case 2
        dNdxi1 = ((xi(2) - 1)*(459*xi(1)^2*xi(2)^2 + 108*xi(1)^2*xi(2) - 255*xi(1)^2 + 234*xi(1)*
        dNdxi2 = ((xi(1) + 1)*(459*xi(1)^2*xi(2)^2 - 234*xi(1)^2*xi(2) - 121*xi(1)^2 - 108*xi(1)*
      case 3
        dNdxi1 = ((xi(2) + 1)*(- 459*xi(1)^2*xi(2)^2 + 108*xi(1)^2*xi(2) + 255*xi(1)^2 - 234*xi(1
        dNdxi2 = ((xi(1) + 1)*(- 459*xi(1)^2*xi(2)^2 - 234*xi(1)^2*xi(2) + 121*xi(1)^2 + 108*xi(1
      case 4
        dNdxi1 = ((xi(2) + 1)*(459*xi(1)^2*xi(2)^2 - 108*xi(1)^2*xi(2) - 255*xi(1)^2 - 234*xi(1)*
        dNdxi2 = ((xi(1) - 1)*(459*xi(1)^2*xi(2)^2 + 234*xi(1)^2*xi(2) - 121*xi(1)^2 + 108*xi(1)*
      case 5
        dNdxi1 = - (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(2) - 1))/64 - (a12*(3*xi(1)
        dNdxi2 = - (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(1) + 1))/64 - (a12*(3*xi(1)
      case 6
        dNdxi1 = (a12*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(2) - 1)^2*(xi(2) + 1))/64 - (a11*(3*xi(2) -
        dNdxi2 = (a12*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) + 1)*(xi(2) - 1)^2)/64 - (a11*(3*xi(2) -
      case 7
        dNdxi1 = (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)^2*(xi(2) + 1))/64 + (a12*(3*xi(1) -
        dNdxi2 = (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)^2)/64 + (a12*(3*xi(1) -
      case 8
        dNdxi1 = (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(2) + 1))/64 - (a12*(3*xi(1) -
        dNdxi2 = (a11*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(1) + 1))/64 - (a12*(3*xi(1) -
      case 9
        dNdxi1 = - (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(2) - 1))/64 - (a22*(3*xi(1)
        dNdxi2 = - (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(1) + 1))/64 - (a22*(3*xi(1)
      case 10
        dNdxi1 = (a22*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(2) - 1)^2*(xi(2) + 1))/64 - (a21*(3*xi(2) -
        dNdxi2 = (a22*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) + 1)*(xi(2) - 1)^2)/64 - (a21*(3*xi(2) -
      case 11
        dNdxi1 = (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) + 1)^2*(xi(2) + 1))/64 + (a22*(3*xi(1) -
        dNdxi2 = (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)^2)/64 + (a22*(3*xi(1) -
      case 12
        dNdxi1 = (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(2) + 1))/64 - (a22*(3*xi(1) -
        dNdxi2 = (a21*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(1) - 1)^2*(xi(1) + 1))/64 - (a22*(3*xi(1) -
      case 13
        dNdxi1 = -(81*(3*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(- 9*xi(1)^2 + 2*xi(1) + 3))/256;
        dNdxi2 = -(81*(3*xi(1) - 1)*(xi(1) - 1)*(xi(1) + 1)*(- 9*xi(2)^2 + 2*xi(2) + 3))/256;
      case 14
        dNdxi1 = -(81*(3*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(9*xi(1)^2 + 2*xi(1) - 3))/256;
        dNdxi2 = (81*(3*xi(1) + 1)*(xi(1) - 1)*(xi(1) + 1)*(- 9*xi(2)^2 + 2*xi(2) + 3))/256;
      case 15
        dNdxi1 = (81*(3*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(9*xi(1)^2 + 2*xi(1) - 3))/256;
        dNdxi2 = (81*(3*xi(1) + 1)*(xi(1) - 1)*(xi(1) + 1)*(9*xi(2)^2 + 2*xi(2) - 3))/256;
      case 16
        dNdxi1 = (81*(3*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(- 9*xi(1)^2 + 2*xi(1) + 3))/256;
        dNdxi2 = -(81*(3*xi(1) - 1)*(xi(1) - 1)*(xi(1) + 1)*(9*xi(2)^2 + 2*xi(2) - 3))/256;
    end

    if( i_eval==2 )
      vBase = aInvJac(:,1).*dNdxi1 + aInvJac(:,2).*dNdxi2;
    else
      vBase = aInvJac(:,3).*dNdxi1 + aInvJac(:,4).*dNdxi2;
    end
```

```
  case {22,23,32,33}   % Evaluation of second order derivatives.
    error('sf_quad_H3: second order derivative evaluation not supported.')

  otherwise
    vBase = 0;

end
```

11.54 ellib/sf_quad_Q1

SF_QUAD_Q1 Bilinear conforming shape function for quadrilaterals (Q1).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_QUAD_Q1(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming bilinear Q1 shape functions on quadrilaterals with values defined in the nodes. XI is [-1..1]^2 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  2             Number of space dimensions
n_vert      scalar:  4             Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                             .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag1, sf_quad_Q1nc

Code listing

```
nLDof = [4 0 0 0];
xLDof = [-1  1 1 -1; ...
          -1 -1 1  1];
sfun  = 'sf_quad_Q1';


switch i_eval   % Evaluation type flag.

  case 1   % Evaluation of function values.

    switch i_dof   % Basis function to evaluate.

      case 1
        vBase = (1-xi(1))*(1-xi(2))/4;
      case 2
        vBase = (1+xi(1))*(1-xi(2))/4;
      case 3
        vBase = (1+xi(1))*(1+xi(2))/4;
      case 4
        vBase = (1-xi(1))*(1+xi(2))/4;
    end

  case {2,3}   % Evaluation of first order derivatives.

    switch i_dof   % Basis function to evaluate.

      case 1
        dNdxi1 = -(1-xi(2))/4;
        dNdxi2 = -(1-xi(1))/4;
      case 2
        dNdxi1 =  (1-xi(2))/4;
        dNdxi2 = -(1+xi(1))/4;
      case 3
        dNdxi1 =  (1+xi(2))/4;
        dNdxi2 =  (1+xi(1))/4;
      case 4
        dNdxi1 = -(1+xi(2))/4;
        dNdxi2 =  (1-xi(1))/4;
    end

    if     ( i_eval==2 )   % x-derivative.
```

```
        vBase = aInvJac(:,1)*dNdxi1+aInvJac(:,2)*dNdxi2;

      elseif ( i_eval==3 )   % y-derivative.

        vBase = aInvJac(:,3)*dNdxi1+aInvJac(:,4)*dNdxi2;
      end

    case {22,23,32,33}   % Evaluation of second order derivatives.

      if( any(abs([aInvJac(:,2);aInvJac(:,3)])>eps*1e2) )
        warning('sf_quad_Q1: 2nd derivatives for non-rectangular cells shapes not supported.')
      end

      switch i_dof   % Basis function to evaluate.

        case {1,3}
          d2Ndxi12    = 0;
          d2Ndxi1dxi2 = 1/4;
          d2Ndxi2dxi1 = 1/4;
          d2Ndxi22    = 0;
        case {2,4}
          d2Ndxi12    = 0;
          d2Ndxi1dxi2 = -1/4;
          d2Ndxi2dxi1 = -1/4;
          d2Ndxi22    = 0;
      end

      if ( i_eval==22 )   % xx-derivative.

        vBase  = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi12    + aInvJac(:,2)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi2dxi1 + aInvJac(:,2)*d2Ndxi22
);

      elseif ( i_eval==23 )   % xy-derivative.

        vBase  = aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi12    + aInvJac(:,2)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi2dxi1 + aInvJac(:,2)*d2Ndxi22
);

      elseif ( i_eval==32 )   % yx-derivative.

        vBase  = aInvJac(:,1).*( aInvJac(:,3)*d2Ndxi12    + aInvJac(:,4)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,2).*( aInvJac(:,3)*d2Ndxi2dxi1 + aInvJac(:,4)*d2Ndxi22
);

      elseif ( i_eval==33 )   % yy-derivative.

        vBase  = aInvJac(:,3).*( aInvJac(:,3)*d2Ndxi12    + aInvJac(:,4)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,4).*( aInvJac(:,3)*d2Ndxi2dxi1 + aInvJac(:,4)*d2Ndxi22
);
      end
```

```
    otherwise
      vBase = 0;

  end
```

11.55 ellib/sf_quad_Q1nc

SF_QUAD_Q1NC Bilinear nonconforming shape function for quadrilaterals (Q1∼).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_QUAD_Q1NC(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates nonconforming rotated bilinear Q1∼ shape functions on quadrilaterals with values defined on the edge midpoints. XI is
[-1..1]^2 reference coordinates.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  2             Number of space dimensions
n_vert      scalar:  4             Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                            .
Output      Value/[Size]           Description
-----------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag1, sf_quad_Q1

Code listing

```
nLDof = [0 4 0 0];
xLDof = [ 0  1 0 -1; ...
         -1  0 1  0];
sfun  = 'sf_quad_Q1nc';


switch i_eval   % Evaluation type flag.

  case 1   % Evaluation of function values.

    switch i_dof   % Basis function to evaluate.

      case 1
        vBase = (-xi(1)^2+xi(2)^2-2*xi(2)+1)/4;
      case 2
        vBase = ( xi(1)^2-xi(2)^2+2*xi(1)+1)/4;
      case 3
        vBase = (-xi(1)^2+xi(2)^2+2*xi(2)+1)/4;
      case 4
        vBase = ( xi(1)^2-xi(2)^2-2*xi(1)+1)/4;
    end

  case {2,3}   % Evaluation of first order derivatives.

    switch i_dof   % Basis function to evaluate.

      case 1
        dNdxi1 = -xi(1)/2;
        dNdxi2 = (xi(2)-1)/2;
      case 2
        dNdxi1 = (xi(1)+1)/2;
        dNdxi2 = -xi(2)/2;
      case 3
        dNdxi1 = -xi(1)/2;
        dNdxi2 = (xi(2)+1)/2;
      case 4
        dNdxi1 = (xi(1)-1)/2;
        dNdxi2 = -xi(2)/2;
    end

    if     ( i_eval==2 )   % x-derivative.

      vBase = aInvJac(:,1)*dNdxi1+aInvJac(:,2)*dNdxi2;

    elseif ( i_eval==3 )   % y-derivative.

      vBase = aInvJac(:,3)*dNdxi1+aInvJac(:,4)*dNdxi2;
    end

  case {22,23,32,33}   % Evaluation of second order derivatives.
```

```
      if( any(abs([aInvJac(:,2);aInvJac(:,3)])>eps*1e2) )
        warning('sf_quad_Q1nc: 2nd derivatives for non-rectangular cells shapes not supported.')
      end

      switch i_dof   % Basis function to evaluate.

        case {1,3}
          d2Ndxi12    = -1/2;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi22    = 1/2;
        case {2,4}
          d2Ndxi12    = 1/2;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi22    = -1/2;
      end

      if ( i_eval==22 )   % xx-derivative.

        vBase  = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi12    + aInvJac(:,2)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi2dxi1 + aInvJac(:,2)*d2Ndxi22
    );

      elseif ( i_eval==23 )   % xy-derivative.

        vBase  = aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi12    + aInvJac(:,2)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi2dxi1 + aInvJac(:,2)*d2Ndxi22
    );

      elseif ( i_eval==32 )   % yx-derivative.

        vBase  = aInvJac(:,1).*( aInvJac(:,3)*d2Ndxi12    + aInvJac(:,4)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,2).*( aInvJac(:,3)*d2Ndxi2dxi1 + aInvJac(:,4)*d2Ndxi22
    );

      elseif ( i_eval==33 )   % yy-derivative.

        vBase  = aInvJac(:,3).*( aInvJac(:,3)*d2Ndxi12    + aInvJac(:,4)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,4).*( aInvJac(:,3)*d2Ndxi2dxi1 + aInvJac(:,4)*d2Ndxi22
    );
      end

    otherwise
      vBase = 0;

  end
```

11.56 ellib/sf_quad_Q2

SF_QUAD_Q2 Biquadratic conforming shape function for quadrilaterals (Q2).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_QUAD_Q2(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming biquadratic Q2 shape functions on quadrilaterals with values defined in the nodes, edges, and cell center. XI is
[-1..1]^2 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  2             Number of space dimensions
n_vert      scalar:  4             Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag2

Code listing

```
nLDof = [4 4 0 1];
xLDof = [-1  1 1 -1  0 1 0 -1 0; ...
          -1 -1 1  1 -1 0 1  0 0];
sfun  = 'sf_quad_Q2';


switch i_eval   % Evaluation type flag.
```

```
case 1   % Evaluation of function values.

  switch i_dof   % Basis function to evaluate.
    case 1
      vBase =  (1-xi(1))*(1-xi(2))*xi(1)*xi(2)/4;
    case 2
      vBase = -(1+xi(1))*(1-xi(2))*xi(1)*xi(2)/4;
    case 3
      vBase =  (1+xi(1))*(1+xi(2))*xi(1)*xi(2)/4;
    case 4
      vBase = -(1-xi(1))*(1+xi(2))*xi(1)*xi(2)/4;
    case 5
      vBase = -(1-xi(1)*xi(1))*(1-xi(2))*xi(2)/2;
    case 6
      vBase =  (1+xi(1))*(1-xi(2)*xi(2))*xi(1)/2;
    case 7
      vBase =  (1-xi(1)*xi(1))*(1+xi(2))*xi(2)/2;
    case 8
      vBase = -(1-xi(1))*(1-xi(2)*xi(2))*xi(1)/2;
    case 9
      vBase =  (1-xi(1)*xi(1))*(1-xi(2)*xi(2));
  end

case {2,3}   % Evaluation of first order derivatives.

  switch i_dof   % Basis function to evaluate.
    case 1
      dNdxi1 =  (1-2*xi(1))*(1-xi(2))*xi(2)/4;
      dNdxi2 =  (1-xi(1))*(1-2*xi(2))*xi(1)/4;
    case 2
      dNdxi1 = -(1+2*xi(1))*(1-xi(2))*xi(2)/4;
      dNdxi2 = -(1+xi(1))*(1-2*xi(2))*xi(1)/4;
    case 3
      dNdxi1 =  (1+2*xi(1))*(1+xi(2))*xi(2)/4;
      dNdxi2 =  (1+xi(1))*(1+2*xi(2))*xi(1)/4;
    case 4
      dNdxi1 = -(1-2*xi(1))*(1+xi(2))*xi(2)/4;
      dNdxi2 = -(1-xi(1))*(1+2*xi(2))*xi(1)/4;
    case 5
      dNdxi1 =  (1-xi(2))*xi(1)*xi(2);
      dNdxi2 = -(1-xi(1)*xi(1))*(1-2*xi(2))/2;
    case 6
      dNdxi1 =  (1+2*xi(1))*(1-xi(2)*xi(2))/2;
      dNdxi2 = -(1+xi(1))*xi(1)*xi(2);
    case 7
      dNdxi1 = -(1+xi(2))*xi(1)*xi(2);
      dNdxi2 =  (1-xi(1)*xi(1))*(1+2*xi(2))/2;
    case 8
      dNdxi1 = -(1-2*xi(1))*(1-xi(2)*xi(2))/2;
      dNdxi2 =  (1-xi(1))*xi(1)*xi(2);
    case 9
```

```
        dNdxi1 = -2*(1-xi(2)*xi(2))*xi(1);
        dNdxi2 = -2*(1-xi(1)*xi(1))*xi(2);
    end

    if     ( i_eval==2 )   % x-derivative.

      vBase = aInvJac(:,1)*dNdxi1+aInvJac(:,2)*dNdxi2;

    elseif ( i_eval==3 )   % y-derivative.

      vBase = aInvJac(:,3)*dNdxi1+aInvJac(:,4)*dNdxi2;
    end

  case {22,23,32,33}

    if( any(abs([aInvJac(:,2);aInvJac(:,3)])>eps*1e2) )
      warning('sf_quad_Q2: 2nd derivatives for non-rectangular cells shapes not supported.')
    end

    switch i_dof   % Basis function to evaluate.
      case 1
        d2Ndxi12    = xi(2)^2/2 - xi(2)/2;
        d2Ndxi1dxi2 = ((2*xi(1) - 1)*(2*xi(2) - 1))/4;
        d2Ndxi2dxi1 = ((2*xi(1) - 1)*(2*xi(2) - 1))/4;
        d2Ndxi22    = xi(1)^2/2 - xi(1)/2;
      case 2
        d2Ndxi12    = xi(2)^2/2 - xi(2)/2;
        d2Ndxi1dxi2 = ((2*xi(1) + 1)*(2*xi(2) - 1))/4;
        d2Ndxi2dxi1 = ((2*xi(1) + 1)*(2*xi(2) - 1))/4;
        d2Ndxi22    = xi(1)^2/2 + xi(1)/2;
      case 3
        d2Ndxi12    = xi(2)^2/2 + xi(2)/2;
        d2Ndxi1dxi2 = ((2*xi(1) + 1)*(2*xi(2) + 1))/4;
        d2Ndxi2dxi1 = ((2*xi(1) + 1)*(2*xi(2) + 1))/4;
        d2Ndxi22    = xi(1)^2/2 + xi(1)/2;
      case 4
        d2Ndxi12    = xi(2)^2/2 + xi(2)/2;
        d2Ndxi1dxi2 = ((2*xi(1) - 1)*(2*xi(2) + 1))/4;
        d2Ndxi2dxi1 = ((2*xi(1) - 1)*(2*xi(2) + 1))/4;
        d2Ndxi22    = xi(1)^2/2 - xi(1)/2;
      case 5
        d2Ndxi12    = xi(2) - xi(2)^2;
        d2Ndxi1dxi2 = xi(1) - 2*xi(1)*xi(2);
        d2Ndxi2dxi1 = xi(1) - 2*xi(1)*xi(2);
        d2Ndxi22    = 1 - xi(1)^2;
      case 6
        d2Ndxi12    = 1 - xi(2)^2;
        d2Ndxi1dxi2 = -xi(2)*(2*xi(1) + 1);
        d2Ndxi2dxi1 = -xi(2)*(2*xi(1) + 1);
        d2Ndxi22    = - xi(1)^2 - xi(1);
      case 7
        d2Ndxi12    = - xi(2)^2 - xi(2);
```

```
          d2Ndxi1dxi2 = -xi(1)*(2*xi(2) + 1);
          d2Ndxi2dxi1 = -xi(1)*(2*xi(2) + 1);
          d2Ndxi22    = 1 - xi(1)^2;
        case 8
          d2Ndxi12    = 1 - xi(2)^2;
          d2Ndxi1dxi2 = xi(2) - 2*xi(1)*xi(2);
          d2Ndxi2dxi1 = xi(2) - 2*xi(1)*xi(2);
          d2Ndxi22    = xi(1) - xi(1)^2;
        case 9
          d2Ndxi12    = 2*xi(2)^2 - 2;
          d2Ndxi1dxi2 = 4*xi(1)*xi(2);
          d2Ndxi2dxi1 = 4*xi(1)*xi(2);
          d2Ndxi22    = 2*xi(1)^2 - 2;
      end

      if ( i_eval==22 )   % xx-derivative.

        vBase  = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi12    + aInvJac(:,2)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi2dxi1 + aInvJac(:,2)*d2Ndxi22
);

      elseif ( i_eval==23 )   % xy-derivative.

        vBase  = aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi12    + aInvJac(:,2)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi2dxi1 + aInvJac(:,2)*d2Ndxi22
);

      elseif ( i_eval==32 )   % yx-derivative.

        vBase  = aInvJac(:,1).*( aInvJac(:,3)*d2Ndxi12    + aInvJac(:,4)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,2).*( aInvJac(:,3)*d2Ndxi2dxi1 + aInvJac(:,4)*d2Ndxi22
);

      elseif ( i_eval==33 )   % yy-derivative.

        vBase  = aInvJac(:,3).*( aInvJac(:,3)*d2Ndxi12    + aInvJac(:,4)*d2Ndxi1dxi2 ) + ...
                 aInvJac(:,4).*( aInvJac(:,3)*d2Ndxi2dxi1 + aInvJac(:,4)*d2Ndxi22
);
      end

    otherwise
      vBase = 0;

  end
```

11.57 ellib/sf_quad_Q3

SF_QUAD_Q3 Bicubic conforming shape function for quadrilaterals (Q3).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_QUAD_Q3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming bicubic Q3 shape functions on quadrilaterals with values defined in the nodes, edges, and cell center. XI is
[-1..1]^2 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  2             Number of space dimensions
n_vert      scalar:  4             Number of vertices per cell
i_dof       scalar: 1-16           Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag3, sf_quad_H3

Code listing

```
nLDof = [4 8 0 4];
xLDof = [-1  1 1 -1 -1/3  1   1/3 -1    1/3 1   -1/3 -1   -1/3  1/3 1/3 -1/3;
         -1 -1 1  1 -1   -1/3 1    1/3 -1   1/3  1   -1/3 -1/3 -1/3 1/3  1/3];
sfun  = 'sf_quad_Q3';


switch i_eval

  case 1

    switch i_dof
      case 1
```

```
          vBase = (3*xi(1) - 1)*((3*xi(1))/16 + 1/16)*(3*xi(2) - 1)*((3*xi(2))/16 + 1/16)*(xi(1) -
        case 2
          vBase = -(3*xi(1) - 1)*((3*xi(1))/16 + 1/16)*(3*xi(2) - 1)*((3*xi(2))/16 + 1/16)*(xi(1) +
        case 3
          vBase = (3*xi(1) - 1)*((3*xi(1))/16 + 1/16)*(3*xi(2) - 1)*((3*xi(2))/16 + 1/16)*(xi(1) +
        case 4
          vBase = -(3*xi(1) - 1)*((3*xi(1))/16 + 1/16)*(3*xi(2) - 1)*((3*xi(2))/16 + 1/16)*(xi(1) -
        case 5
          vBase = -(3*xi(1) - 1)*((9*xi(1))/16 + 9/16)*(3*xi(2) - 1)*((3*xi(2))/16 + 1/16)*(xi(1) -
        case 6
          vBase = (3*xi(1) - 1)*((3*xi(1))/16 + 1/16)*(3*xi(2) - 1)*((9*xi(2))/16 + 9/16)*(xi(1) +
        case 7
          vBase = -(3*xi(1) + 1)*((9*xi(1))/16 + 9/16)*(3*xi(2) - 1)*((3*xi(2))/16 + 1/16)*(xi(1) -
        case 8
          vBase = (3*xi(1) - 1)*((3*xi(1))/16 + 1/16)*(3*xi(2) + 1)*((9*xi(2))/16 + 9/16)*(xi(1) -
        case 9
          vBase = (3*xi(1) + 1)*((9*xi(1))/16 + 9/16)*(3*xi(2) - 1)*((3*xi(2))/16 + 1/16)*(xi(1) -
        case 10
          vBase = -(3*xi(1) - 1)*((3*xi(1))/16 + 1/16)*(3*xi(2) + 1)*((9*xi(2))/16 + 9/16)*(xi(1) +
        case 11
          vBase = (3*xi(1) - 1)*((9*xi(1))/16 + 9/16)*(3*xi(2) - 1)*((3*xi(2))/16 + 1/16)*(xi(1) -
        case 12
          vBase = -(3*xi(1) - 1)*((3*xi(1))/16 + 1/16)*(3*xi(2) - 1)*((9*xi(2))/16 + 9/16)*(xi(1) -
        case 13
          vBase = (3*xi(1) - 1)*((9*xi(1))/16 + 9/16)*(3*xi(2) - 1)*((9*xi(2))/16 + 9/16)*(xi(1) -
        case 14
          vBase = -(3*xi(1) + 1)*((9*xi(1))/16 + 9/16)*(3*xi(2) - 1)*((9*xi(2))/16 + 9/16)*(xi(1) -
        case 15
          vBase = (3*xi(1) + 1)*((9*xi(1))/16 + 9/16)*(3*xi(2) + 1)*((9*xi(2))/16 + 9/16)*(xi(1) -
        case 16
          vBase = -(3*xi(1) - 1)*((9*xi(1))/16 + 9/16)*(3*xi(2) + 1)*((9*xi(2))/16 + 9/16)*(xi(1) -
      end

    case {2,3}

      switch i_dof
        case 1
          dNdxi1 = -((3*xi(2) - 1)*(3*xi(2) + 1)*(xi(2) - 1)*(- 27*xi(1)^2 + 18*xi(1) + 1))/256;
          dNdxi2 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) - 1)*(- 27*xi(2)^2 + 18*xi(2) + 1))/256;
        case 2
          dNdxi1 = -((3*xi(2) - 1)*(3*xi(2) + 1)*(xi(2) - 1)*(27*xi(1)^2 + 18*xi(1) - 1))/256;
          dNdxi2 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) + 1)*(- 27*xi(2)^2 + 18*xi(2) + 1))/256;
        case 3
          dNdxi1 = ((3*xi(2) - 1)*(3*xi(2) + 1)*(xi(2) + 1)*(27*xi(1)^2 + 18*xi(1) - 1))/256;
          dNdxi2 = ((3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) + 1)*(27*xi(2)^2 + 18*xi(2) - 1))/256;
        case 4
          dNdxi1 = ((3*xi(2) - 1)*(3*xi(2) + 1)*(xi(2) + 1)*(- 27*xi(1)^2 + 18*xi(1) + 1))/256;
          dNdxi2 = -((3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) - 1)*(27*xi(2)^2 + 18*xi(2) - 1))/256;
        case 5
          dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(2) - 1)*(- 9*xi(1)^2 + 2*xi(1) + 3))/256;
          dNdxi2 = (9*(3*xi(1) - 1)*(xi(1) - 1)*(xi(1) + 1)*(- 27*xi(2)^2 + 18*xi(2) + 1))/256;
        case 6
```

```
          dNdxi1 = (9*(3*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(27*xi(1)^2 + 18*xi(1) - 1))/256;
          dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) + 1)*(- 9*xi(2)^2 + 2*xi(2) + 3))/256;
        case 7
          dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(2) + 1)*(9*xi(1)^2 + 2*xi(1) - 3))/256;
          dNdxi2 = -(9*(3*xi(1) + 1)*(xi(1) - 1)*(xi(1) + 1)*(27*xi(2)^2 + 18*xi(2) - 1))/256;
        case 8
          dNdxi1 = -(9*(3*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(- 27*xi(1)^2 + 18*xi(1) + 1))/256;
          dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) - 1)*(9*xi(2)^2 + 2*xi(2) - 3))/256;
        case 9
          dNdxi1 = (9*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(2) - 1)*(9*xi(1)^2 + 2*xi(1) - 3))/256;
          dNdxi2 = -(9*(3*xi(1) + 1)*(xi(1) - 1)*(xi(1) + 1)*(- 27*xi(2)^2 + 18*xi(2) + 1))/256;
        case 10
          dNdxi1 = -(9*(3*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(27*xi(1)^2 + 18*xi(1) - 1))/256;
          dNdxi2 = -(9*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) + 1)*(9*xi(2)^2 + 2*xi(2) - 3))/256;
        case 11
          dNdxi1 = -(9*(3*xi(2) - 1)*(3*xi(2) + 1)*(xi(2) + 1)*(- 9*xi(1)^2 + 2*xi(1) + 3))/256;
          dNdxi2 = (9*(3*xi(1) - 1)*(xi(1) - 1)*(xi(1) + 1)*(27*xi(2)^2 + 18*xi(2) - 1))/256;
        case 12
          dNdxi1 = (9*(3*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(- 27*xi(1)^2 + 18*xi(1) + 1))/256;
          dNdxi2 = (9*(3*xi(1) - 1)*(3*xi(1) + 1)*(xi(1) - 1)*(- 9*xi(2)^2 + 2*xi(2) + 3))/256;
        case 13
          dNdxi1 = -(81*(3*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(- 9*xi(1)^2 + 2*xi(1) + 3))/256;
          dNdxi2 = -(81*(3*xi(1) - 1)*(xi(1) - 1)*(xi(1) + 1)*(- 9*xi(2)^2 + 2*xi(2) + 3))/256;
        case 14
          dNdxi1 = -(81*(3*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(9*xi(1)^2 + 2*xi(1) - 3))/256;
          dNdxi2 = (81*(3*xi(1) + 1)*(xi(1) - 1)*(xi(1) + 1)*(- 9*xi(2)^2 + 2*xi(2) + 3))/256;
        case 15
          dNdxi1 = (81*(3*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(9*xi(1)^2 + 2*xi(1) - 3))/256;
          dNdxi2 = (81*(3*xi(1) + 1)*(xi(1) - 1)*(xi(1) + 1)*(9*xi(2)^2 + 2*xi(2) - 3))/256;
        case 16
          dNdxi1 = (81*(3*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(- 9*xi(1)^2 + 2*xi(1) + 3))/256;
          dNdxi2 = -(81*(3*xi(1) - 1)*(xi(1) - 1)*(xi(1) + 1)*(9*xi(2)^2 + 2*xi(2) - 3))/256;
      end

      if( i_eval==2 )
        vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2;
      else
        vBase = aInvJac(:,3)*dNdxi1 + aInvJac(:,4)*dNdxi2;
      end

    case {22,23,32,33}   % Evaluation of second order derivatives.
      error('sf_quad_Q3: second order derivative evaluation not supported.')

    otherwise
      vBase = 0;

  end
```

11.58 ellib/sf_quad_Q4

SF_QUAD_Q4 Biquartic conforming shape function for quadrilaterals (Q4).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_QUAD_Q4(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming biquartic Q4 shape functions on quadrilaterals with values defined in the nodes, edges, and cell center. XI is
[-1..1]^2 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  2             Number of space dimensions
n_vert      scalar:  4             Number of vertices per cell
i_dof       scalar: 1-25           Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_quad_Q1

Code listing

```
 nLDof = [4 12 0 9];
 xLDof = [-1  1 1 -1  -1/2  1   1/2 -1     0  1 0 -1   1/2 1   -1/2 -1    -1/2  1/2 1/2 -1/2
0   1/2 0   -1/2 0;
          -1 -1 1  1  -1   -1/2 1    1/2  -1  0 1  0  -1   1/2  1   -1/2  -1/2 -1/2 1/2
1/2 -1/2 0   1/2  0   0];
 sfun  = 'sf_quad_Q4';
```

```
switch i_eval

  case 1

    switch i_dof
      case 1
        vBase = (xi(1)*xi(2)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(
      case 2
        vBase = (xi(1)*xi(2)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(
      case 3
        vBase = (xi(1)*xi(2)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(
      case 4
        vBase = (xi(1)*xi(2)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(
      case 5
        vBase = -(2*xi(1)*xi(2)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^2 +
      case 6
        vBase = -(2*xi(1)*xi(2)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*
      case 7
        vBase = -(2*xi(1)*xi(2)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*
      case 8
        vBase = -(2*xi(1)*xi(2)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*
      case 9
        vBase = -(xi(2)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1))/6;
      case 10
        vBase = -(xi(1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1))/6;
      case 11
        vBase = -(xi(2)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 4*xi(2)^3 - 4*xi(2)^2 + xi(2) + 1))/6;
      case 12
        vBase = -(xi(1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1))/6;
      case 13
        vBase = -(2*xi(1)*xi(2)*(- 4*xi(2)^3 + 4*xi(2)^2 + xi(2) - 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*
      case 14
        vBase = -(2*xi(1)*xi(2)*(- 4*xi(1)^3 - 4*xi(1)^2 + xi(1) + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*
      case 15
        vBase = -(2*xi(1)*xi(2)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 4*xi(2)^3 + 4*xi(2)^2 +
      case 16
        vBase = -(2*xi(1)*xi(2)*(- 4*xi(1)^3 + 4*xi(1)^2 + xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*
      case 17
        vBase = (16*xi(1)*xi(2)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*
      case 18
        vBase = (16*xi(1)*xi(2)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*
      case 19
        vBase = (16*xi(1)*xi(2)*(- 2*xi(1)^3 - xi(1)^2 + 2*xi(1) + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*
      case 20
        vBase = (16*xi(1)*xi(2)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*
      case 21
        vBase = (4*xi(2)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 2*xi(2)^3 + xi(2)^2 + 2*xi(2) - 1))/3;
      case 22
        vBase = (4*xi(1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(1)^3 - xi(1)^2 + 2*xi(1) + 1))/3;
      case 23
        vBase = (4*xi(2)*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 2*xi(2)^3 - xi(2)^2 + 2*xi(2) + 1))/3;
```

```
        case 24
          vBase = (4*xi(1)*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 2*xi(1)^3 + xi(1)^2 + 2*xi(1) - 1))/3;
        case 25
          vBase = (4*xi(1)^4 - 5*xi(1)^2 + 1)*(4*xi(2)^4 - 5*xi(2)^2 + 1);
      end

    case {2,3}

      switch i_dof
        case 1
          dNdxi1 = -(xi(2)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) - 1)*(- 4*xi(1)^2 + 2*x
          dNdxi2 = -(xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(xi(1) - 1)*(- 4*xi(2)^2 + 2*x
        case 2
          dNdxi1 = (xi(2)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) - 1)*(4*xi(1)^2 + 2*xi(1
          dNdxi2 = -(xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(xi(1) + 1)*(- 4*xi(2)^2 + 2*x
        case 3
          dNdxi1 = (xi(2)*(4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) + 1)*(4*xi(1)^2 + 2*xi(1
          dNdxi2 = (xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(xi(1) + 1)*(4*xi(2)^2 + 2*xi(2
        case 4
          dNdxi1 = -(xi(2)*(4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) + 1)*(- 4*xi(1)^2 + 2*x
          dNdxi2 = (xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(xi(1) - 1)*(4*xi(2)^2 + 2*xi(2
        case 5
          dNdxi1 = (2*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) - 1)*(- 8*xi(1)^3 + 3*xi(1)^2 + 4*xi
          dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(4*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(- 4*xi(2)^2 + 2*xi
        case 6
          dNdxi1 = -(2*xi(2)*(4*xi(1) + 1)*(2*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(4*xi(1)^2 + 2*xi(
          dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(xi(1) + 1)*(- 8*xi(2)^3 + 3*xi(2)^2 + 4*xi
        case 7
          dNdxi1 = (2*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) + 1)*(- 8*xi(1)^3 - 3*xi(1)^2 + 4*xi
          dNdxi2 = -(2*xi(1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(4*xi(2)^2 + 2*xi(
        case 8
          dNdxi1 = (2*xi(2)*(4*xi(1) - 1)*(2*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(- 4*xi(1)^2 + 2*xi
          dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(xi(1) - 1)*(- 8*xi(2)^3 - 3*xi(2)^2 + 4*xi
        case 9
          dNdxi1 = (xi(1)*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(1)^2 - 5)*(xi(2) - 1))/3;
          dNdxi2 = -((2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(- 4*xi(2)^2
        case 10
          dNdxi1 = ((4*xi(1) + 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(4*xi(1)^2 +
          dNdxi2 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(8*xi(2)^2 - 5)*(xi(1) + 1))/3;
        case 11
          dNdxi1 = (xi(1)*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(8*xi(1)^2 - 5)*(xi(2) + 1))/3;
          dNdxi2 = ((2*xi(1) - 1)*(2*xi(1) + 1)*(4*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(4*xi(2)^2 +
        case 12
          dNdxi1 = -((4*xi(1) - 1)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(- 4*xi(1)^2
          dNdxi2 = (xi(1)*xi(2)*(2*xi(1) - 1)*(2*xi(1) + 1)*(8*xi(2)^2 - 5)*(xi(1) - 1))/3;
        case 13
          dNdxi1 = (2*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) - 1)*(- 8*xi(1)^3 - 3*xi(1)^2 + 4*xi
          dNdxi2 = (2*xi(1)*(2*xi(1) + 1)*(4*xi(2) - 1)*(xi(1) - 1)*(xi(1) + 1)*(- 4*xi(2)^2 + 2*xi
        case 14
          dNdxi1 = -(2*xi(2)*(4*xi(1) + 1)*(2*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(4*xi(1)^2 + 2*xi(
          dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(xi(1) + 1)*(- 8*xi(2)^3 - 3*xi(2)^2 + 4*xi
        case 15
```

```
          dNdxi1 = (2*xi(2)*(2*xi(2) - 1)*(2*xi(2) + 1)*(xi(2) + 1)*(- 8*xi(1)^3 + 3*xi(1)^2 + 4*xi
          dNdxi2 = -(2*xi(1)*(2*xi(1) - 1)*(4*xi(2) + 1)*(xi(1) - 1)*(xi(1) + 1)*(4*xi(2)^2 + 2*xi(
        case 16
          dNdxi1 = (2*xi(2)*(4*xi(1) - 1)*(2*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(- 4*xi(1)^2 + 2*xi
          dNdxi2 = (2*xi(1)*(2*xi(1) - 1)*(2*xi(1) + 1)*(xi(1) - 1)*(- 8*xi(2)^3 + 3*xi(2)^2 + 4*xi
        case 17
          dNdxi1 = -(16*xi(2)*(2*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(- 8*xi(1)^3 + 3*xi(1)^2 + 4*xi
          dNdxi2 = -(16*xi(1)*(2*xi(1) - 1)*(xi(1) - 1)*(xi(1) + 1)*(- 8*xi(2)^3 + 3*xi(2)^2 + 4*xi
        case 18
          dNdxi1 = -(16*xi(2)*(2*xi(2) - 1)*(xi(2) - 1)*(xi(2) + 1)*(- 8*xi(1)^3 - 3*xi(1)^2 + 4*xi
          dNdxi2 = -(16*xi(1)*(2*xi(1) + 1)*(xi(1) - 1)*(xi(1) + 1)*(- 8*xi(2)^3 + 3*xi(2)^2 + 4*xi
        case 19
          dNdxi1 = -(16*xi(2)*(2*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(- 8*xi(1)^3 - 3*xi(1)^2 + 4*xi
          dNdxi2 = -(16*xi(1)*(2*xi(1) + 1)*(xi(1) - 1)*(xi(1) + 1)*(- 8*xi(2)^3 - 3*xi(2)^2 + 4*xi
        case 20
          dNdxi1 = -(16*xi(2)*(2*xi(2) + 1)*(xi(2) - 1)*(xi(2) + 1)*(- 8*xi(1)^3 + 3*xi(1)^2 + 4*xi
          dNdxi2 = -(16*xi(1)*(2*xi(1) - 1)*(xi(1) - 1)*(xi(1) + 1)*(- 8*xi(2)^3 - 3*xi(2)^2 + 4*xi
        case 21
          dNdxi1 = -(8*xi(1)*xi(2)*(2*xi(2) - 1)*(8*xi(1)^2 - 5)*(xi(2) - 1)*(xi(2) + 1))/3;
          dNdxi2 = (4*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 8*xi(2)^3 + 3*xi(2)^2 + 4*xi(2) - 1))/3;
        case 22
          dNdxi1 = (4*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 8*xi(1)^3 - 3*xi(1)^2 + 4*xi(1) + 1))/3;
          dNdxi2 = -(8*xi(1)*xi(2)*(2*xi(1) + 1)*(8*xi(2)^2 - 5)*(xi(1) - 1)*(xi(1) + 1))/3;
        case 23
          dNdxi1 = -(8*xi(1)*xi(2)*(2*xi(2) + 1)*(8*xi(1)^2 - 5)*(xi(2) - 1)*(xi(2) + 1))/3;
          dNdxi2 = (4*(4*xi(1)^4 - 5*xi(1)^2 + 1)*(- 8*xi(2)^3 - 3*xi(2)^2 + 4*xi(2) + 1))/3;
        case 24
          dNdxi1 = (4*(4*xi(2)^4 - 5*xi(2)^2 + 1)*(- 8*xi(1)^3 + 3*xi(1)^2 + 4*xi(1) - 1))/3;
          dNdxi2 = -(8*xi(1)*xi(2)*(2*xi(1) - 1)*(8*xi(2)^2 - 5)*(xi(1) - 1)*(xi(1) + 1))/3;
        case 25
          dNdxi1 = 2*xi(1)*(8*xi(1)^2 - 5)*(4*xi(2)^4 - 5*xi(2)^2 + 1);
          dNdxi2 = 2*xi(2)*(8*xi(2)^2 - 5)*(4*xi(1)^4 - 5*xi(1)^2 + 1);
      end

      if( i_eval==2 )
        vBase = aInvJac(:,1).*dNdxi1 + aInvJac(:,2).*dNdxi2;
      else
        vBase = aInvJac(:,3).*dNdxi1 + aInvJac(:,4).*dNdxi2;
      end

    case {22,23,32,33}   % Evaluation of second order derivatives.
      error('sf_quad_Q4: second order derivative evaluation not supported.')

    otherwise
      vBase = 0;

  end
```

11.59 ellib/sf_quad_Q5

SF_QUAD_Q5 Biquintic conforming shape function for quadrilaterals (Q5).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_QUAD_Q5(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming biquintic Q5 shape functions on quadrilaterals with values defined in the nodes, edges, and cell center. XI is
[-1..1]^2 reference coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar:  2             Number of space dimensions
n_vert      scalar:  4             Number of vertices per cell
i_dof       scalar: 1-36           Local basis function to evaluate
xi          [n_sdim]               Local coordinates of evaluation point
aInvJac     [n,n_sdim*n_sdim]      Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_quad_Q1

Code listing

```
 nLDof = [4 16 0 16];
 xLDof = [-1  1 1 -1  -3/5  1   3/5 -1    -1/5  1   1/5 -1     1/5  1   -1/5 -1
3/5  1   -3/5 -1    -3/5  3/5 3/5 -3/5  -1/5  3/5 1/5 -3/5   1/5 3/5 -1/5 -3/5  -1/5
1/5 1/5 -1/5;
          -1 -1 1  1  -1   -3/5 1    3/5  -1   -1/5 1    1/5  -1    1/5  1   -1/5
-1    3/5  1   -3/5  -3/5 -3/5 3/5  3/5  -3/5 -1/5 3/5  1/5  -3/5 1/5  3/5 -1/5  -1/5 -1/5 1/5
1/5];
```

```
sfun  = 'sf_quad_Q5';


switch i_eval

  case 1

    switch i_dof
      case 1
        vBase = ((625*xi(1)^5)/768 - (625*xi(1)^4)/768 - (125*xi(1)^3)/384 + (125*xi(1)^2)/384 +
      case 2
        vBase = -((625*xi(1)^5)/768 + (625*xi(1)^4)/768 - (125*xi(1)^3)/384 - (125*xi(1)^2)/384 +
      case 3
        vBase = ((625*xi(1)^5)/768 + (625*xi(1)^4)/768 - (125*xi(1)^3)/384 - (125*xi(1)^2)/384 +
      case 4
        vBase = -((625*xi(1)^5)/768 - (625*xi(1)^4)/768 - (125*xi(1)^3)/384 + (125*xi(1)^2)/384 +
      case 5
        vBase = -((625*xi(2)^5)/768 - (625*xi(2)^4)/768 - (125*xi(2)^3)/384 + (125*xi(2)^2)/384 +
      case 6
        vBase = ((625*xi(1)^5)/768 + (625*xi(1)^4)/768 - (125*xi(1)^3)/384 - (125*xi(1)^2)/384 +
      case 7
        vBase = -((625*xi(2)^5)/768 + (625*xi(2)^4)/768 - (125*xi(2)^3)/384 - (125*xi(2)^2)/384 +
      case 8
        vBase = ((625*xi(1)^5)/768 - (625*xi(1)^4)/768 - (125*xi(1)^3)/384 + (125*xi(1)^2)/384 +
      case 9
        vBase = ((625*xi(2)^5)/768 - (625*xi(2)^4)/768 - (125*xi(2)^3)/384 + (125*xi(2)^2)/384 +
      case 10
        vBase = -((625*xi(1)^5)/768 + (625*xi(1)^4)/768 - (125*xi(1)^3)/384 - (125*xi(1)^2)/384 +
      case 11
        vBase = ((625*xi(2)^5)/768 + (625*xi(2)^4)/768 - (125*xi(2)^3)/384 - (125*xi(2)^2)/384 +
      case 12
        vBase = -((625*xi(1)^5)/768 - (625*xi(1)^4)/768 - (125*xi(1)^3)/384 + (125*xi(1)^2)/384 +
      case 13
        vBase = -((625*xi(2)^5)/768 - (625*xi(2)^4)/768 - (125*xi(2)^3)/384 + (125*xi(2)^2)/384 +
      case 14
        vBase = ((625*xi(1)^5)/768 + (625*xi(1)^4)/768 - (125*xi(1)^3)/384 - (125*xi(1)^2)/384 +
      case 15
        vBase = -((625*xi(2)^5)/768 + (625*xi(2)^4)/768 - (125*xi(2)^3)/384 - (125*xi(2)^2)/384 +
      case 16
        vBase = ((625*xi(1)^5)/768 - (625*xi(1)^4)/768 - (125*xi(1)^3)/384 + (125*xi(1)^2)/384 +
      case 17
        vBase = ((625*xi(2)^5)/768 - (625*xi(2)^4)/768 - (125*xi(2)^3)/384 + (125*xi(2)^2)/384 +
      case 18
        vBase = -((625*xi(1)^5)/768 + (625*xi(1)^4)/768 - (125*xi(1)^3)/384 - (125*xi(1)^2)/384 +
      case 19
        vBase = ((625*xi(2)^5)/768 + (625*xi(2)^4)/768 - (125*xi(2)^3)/384 - (125*xi(2)^2)/384 +
      case 20
        vBase = -((625*xi(1)^5)/768 - (625*xi(1)^4)/768 - (125*xi(1)^3)/384 + (125*xi(1)^2)/384 +
      case 21
        vBase = ((3125*xi(1)^5)/768 - (625*xi(1)^4)/256 - (1625*xi(1)^3)/384 + (325*xi(1)^2)/128
      case 22
        vBase = -((3125*xi(1)^5)/768 + (625*xi(1)^4)/256 - (1625*xi(1)^3)/384 - (325*xi(1)^2)/128
```

```
        case 23
          vBase = ((3125*xi(1)^5)/768 + (625*xi(1)^4)/256 - (1625*xi(1)^3)/384 - (325*xi(1)^2)/128
        case 24
          vBase = -((3125*xi(1)^5)/768 - (625*xi(1)^4)/256 - (1625*xi(1)^3)/384 + (325*xi(1)^2)/128
        case 25
          vBase = -((3125*xi(1)^5)/384 - (625*xi(1)^4)/384 - (2125*xi(1)^3)/192 + (425*xi(1)^2)/192
        case 26
          vBase = ((3125*xi(1)^5)/768 + (625*xi(1)^4)/256 - (1625*xi(1)^3)/384 - (325*xi(1)^2)/128
        case 27
          vBase = -((3125*xi(1)^5)/384 + (625*xi(1)^4)/384 - (2125*xi(1)^3)/192 - (425*xi(1)^2)/192
        case 28
          vBase = ((3125*xi(1)^5)/768 - (625*xi(1)^4)/256 - (1625*xi(1)^3)/384 + (325*xi(1)^2)/128
        case 29
          vBase = ((3125*xi(1)^5)/384 + (625*xi(1)^4)/384 - (2125*xi(1)^3)/192 - (425*xi(1)^2)/192
        case 30
          vBase = -((3125*xi(1)^5)/768 + (625*xi(1)^4)/256 - (1625*xi(1)^3)/384 - (325*xi(1)^2)/128
        case 31
          vBase = ((3125*xi(1)^5)/384 - (625*xi(1)^4)/384 - (2125*xi(1)^3)/192 + (425*xi(1)^2)/192
        case 32
          vBase = -((3125*xi(1)^5)/768 - (625*xi(1)^4)/256 - (1625*xi(1)^3)/384 + (325*xi(1)^2)/128
        case 33
          vBase = ((3125*xi(1)^5)/384 - (625*xi(1)^4)/384 - (2125*xi(1)^3)/192 + (425*xi(1)^2)/192
        case 34
          vBase = -((3125*xi(1)^5)/384 + (625*xi(1)^4)/384 - (2125*xi(1)^3)/192 - (425*xi(1)^2)/192
        case 35
          vBase = ((3125*xi(1)^5)/384 + (625*xi(1)^4)/384 - (2125*xi(1)^3)/192 - (425*xi(1)^2)/192
        case 36
          vBase = -((3125*xi(1)^5)/384 - (625*xi(1)^4)/384 - (2125*xi(1)^3)/192 + (425*xi(1)^2)/192
      end

    case {2,3}

      switch i_dof
        case 1
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*((3125*xi(1
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*((3125*xi(2
        case 2
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*((3125*xi(
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) + 1)*((3125*xi(
        case 3
          dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) + 1)*((3125*xi(1
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) + 1)*((3125*xi(2
        case 4
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) + 1)*((3125*xi(
          dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*((3125*xi(
        case 5
          dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*((15625*xi
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(xi(1) - 1)*(xi(1) + 1)*((3125*xi
        case 6
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(xi(2) - 1)*(xi(2) + 1)*((3125*xi(
          dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) + 1)*((15625*xi(
        case 7
```

```
      dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) + 1)*((15625*xi
      dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((3125*xi
    case 8
      dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((3125*xi(
      dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*((15625*xi(
    case 9
      dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*((15625*xi(
      dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((3125*xi(
    case 10
      dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((3125*xi
      dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) + 1)*((15625*xi
    case 11
      dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) + 1)*((15625*xi(
      dNdxi2 = (25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((3125*xi(
    case 12
      dNdxi1 = -(25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((3125*xi
      dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*((15625*xi
    case 13
      dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*((15625*xi
      dNdxi2 = -(25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((3125*xi
    case 14
      dNdxi1 = (25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((3125*xi(
      dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) + 1)*((15625*xi(
    case 15
      dNdxi1 = -((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) + 1)*((15625*xi
      dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((3125*xi
    case 16
      dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((3125*xi(
      dNdxi2 = ((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*((15625*xi(
    case 17
      dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*((15625*xi(
      dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((3125*xi(
    case 18
      dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((3125*xi
      dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) + 1)*((15625*xi
    case 19
      dNdxi1 = ((5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) + 1)*((15625*xi(
      dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(xi(1) - 1)*(xi(1) + 1)*((3125*xi(
    case 20
      dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(xi(2) - 1)*(xi(2) + 1)*((3125*xi
      dNdxi2 = -((5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*((15625*xi
    case 21
      dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*xi
      dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*xi
    case 22
      dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*x
      dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*x
    case 23
      dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*xi
      dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*xi
    case 24
      dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*x
```

```
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*x
        case 25
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*x
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*x
        case 26
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*xi
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*xi
        case 27
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*x
          dNdxi2 = -(25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*x
        case 28
          dNdxi1 = (25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*xi
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*xi
        case 29
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) - 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*xi
          dNdxi2 = (25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*xi
        case 30
          dNdxi1 = -(25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*x
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*x
        case 31
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) + 1)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*xi
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*xi
        case 32
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*x
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) + 1)*(5*xi(1) - 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*x
        case 33
          dNdxi1 = (25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*xi
          dNdxi2 = (25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*xi
        case 34
          dNdxi1 = -(25*(5*xi(2) - 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*x
          dNdxi2 = -(25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*x
        case 35
          dNdxi1 = (25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*xi
          dNdxi2 = (25*(5*xi(1) + 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*xi
        case 36
          dNdxi1 = -(25*(5*xi(2) + 1)*(5*xi(2) - 3)*(5*xi(2) + 3)*(xi(2) - 1)*(xi(2) + 1)*((15625*x
          dNdxi2 = -(25*(5*xi(1) - 1)*(5*xi(1) - 3)*(5*xi(1) + 3)*(xi(1) - 1)*(xi(1) + 1)*((15625*x
      end

      if( i_eval==2 )
        vBase = aInvJac(:,1).*dNdxi1 + aInvJac(:,2).*dNdxi2;
      else
        vBase = aInvJac(:,3).*dNdxi1 + aInvJac(:,4).*dNdxi2;
      end

    case {22,23,32,33}   % Evaluation of second order derivatives.
      error('sf_quad_Q5: second order derivative evaluation not supported.')

    otherwise
      vBase = 0;

  end
```

11.60 ellib/sf_simp_P1

SF_SIMP_P1 Linear Lagrange shape function for simplices (P1).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_SIMP_P1(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming linear P1 Lagrange shape functions on simplices with values defined in the nodes. XI is Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-4            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim+1]             Local coordinates of evaluation point
aInvJac     [n,n_sdim+1*n_sdim]    Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag1

Code listing

```
nLDof = [n_vert 0 0 0];
xLDof = eye(n_vert);
sfun  = 'sf_simp_P1';
```

```
  switch i_eval    % Evaluation type flag.

    case 1         % Evaluation of function values.

      vBase = xi(i_dof);

    case {2,3,4}   % Evaluation of first derivatives.

      i_col = i_dof+(i_eval-2)*n_vert;

      vBase = aInvJac(:,i_col);

    otherwise
      vBase = 0;

  end
```

11.61 ellib/sf_simp_P1nc

SF_SIMP_P1NC Linear nonconforming shape function for simplices (P-1).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_SIMP_P1NC(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates nonconforming linear P-1 shape functions (Crouzeix-Raviart) on simplices with values defined on the edge/face midpoints. XI is Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 2-3            Number of space dimensions
n_vert      scalar: 2-4            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim+1]             Local coordinates of evaluation point
aInvJac     [n,n_sdim+1*n_sdim]    Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sfnoncon1

Code listing

```
nLDof = [0 0 0 0];
nLDof(n_sdim) = n_sdim+1;
sfun  = 'sf_simp_P1nc';


if ( n_sdim==2 )
  ind   = [3 1 2];
  xLDof = [0.5   0 0.5; ...
           0.5 0.5   0; ...
             0 0.5 0.5];
else
  ind   = [4 3 1 2];
  xLDof = [1/3 1/3   0 1/3; ...
           1/3 1/3 1/3   0; ...
           1/3   0 1/3 1/3; ...
             0 1/3 1/3 1/3];
end

switch i_eval    % Evaluation type flag.

  case 1         % Evaluation of function values.

    vBase = 1-n_sdim*xi(ind(i_dof));

  case {2,3,4}   % Evaluation of first derivatives.

    i_col = ind(i_dof)+(i_eval-2)*n_vert;

    vBase = -n_sdim*aInvJac(:,i_col);

  otherwise
    vBase = 0;

end
```

11.62 ellib/sf_simp_P2

SF_SIMP_P2 Quadratic Lagrange shape function for simplices (P2).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_SIMP_P2(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming quadratic P2 Lagrange shape functions on simplices with values defined in the nodes. XI Barycentric coordinates.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-4            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim+1]             Local coordinates of evaluation point
aInvJac     [n,n_sdim+1*n_sdim]    Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                            .
Output      Value/[Size]           Description
-----------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag2

Code listing

```
switch n_sdim

  case 1
    nLDof = [ 2 0 0 1 ];
    xLDof = [ 1 0 0.5; ...
              0 1 0.5 ];
  case 2
    nLDof = [ 3 3 0 0 ];
    xLDof = [ 1 0 0 0.5 0   0.5; ...
              0 1 0 0.5 0.5 0  ; ...
              0 0 1 0   0.5 0.5 ];
  case 3
    nLDof = [ 4 6 0 0 ];
```

```
      xLDof = [ 1 0 0 0 0.5 0   0.5 0.5 0   0  ; ...
                0 1 0 0 0.5 0.5 0   0   0.5 0  ; ...
                0 0 1 0 0   0.5 0.5 0   0   0.5; ...
                0 0 0 1 0   0   0   0.5 0.5 0.5 ];
  end
  sfun = 'sf_simp_P2';


% Evaluation type flag.
  if( i_eval==1 )    % Evaluation of function values.

      if( i_dof<=n_vert )   % Node dofs.

        vBase = xi(i_dof)*(2*xi(i_dof)-1);

      else   % Edge dofs.

        i = mod(i_dof-n_vert-1,min(n_vert,3))+1;
        j = mod(i,min(n_vert,3))+1;
        if ( i_dof>7 )
          j = 4;
        end

        vBase = 4*xi(i)*xi(j);

      end

  elseif( i_eval>=2 && i_eval<=n_sdim+1 )   % Evaluation of first derivatives.

      if( i_dof<=n_vert )   % Node dofs.

        i_col = i_dof + (i_eval-2)*n_vert;
        dNdxi = 4*xi(i_dof)-1;

        vBase = aInvJac(:,i_col)*dNdxi;

      else   % Edge dofs.

        i = mod(i_dof-n_vert-1,min(n_vert,3))+1;
        j = mod(i,min(n_vert,3))+1;
        if ( i_dof>7 )
          j = 4;
        end
        i_col  = i + (i_eval-2)*n_vert;
        j_col  = j + (i_eval-2)*n_vert;
        dNdxii = 4*xi(j);
        dNdxij = 4*xi(i);

        vBase  = aInvJac(:,i_col)*dNdxii + aInvJac(:,j_col)*dNdxij;

      end
```

```
  elseif( any(i_eval==[22 23 24 32 33 34 42 43 44]) )   % Evaluation of second derivatives.

      i_eval1 = floor(i_eval/10);
      i_eval2 = i_eval - 10*i_eval1;

      if( i_dof<=n_vert )   % Node dofs.

        d2Ndxi2 = 4;

        i_eval = floor(i_eval/10);
        i_col  = i_dof + (i_eval1-2)*n_vert;
        j_col  = i_dof + (i_eval2-2)*n_vert;
        vBase  = aInvJac(:,i_col).*aInvJac(:,j_col)*d2Ndxi2;

      else   % Edge dofs.

        d2Ndxii2    = 0;
        d2Ndxiidxij = 4;
        d2Ndxijdxii = d2Ndxiidxij;
        d2Ndxij2    = 0;

        i = mod(i_dof-n_vert-1,min(n_vert,3))+1;
        j = mod(i,min(n_vert,3))+1;
        if( i_dof>7 )
          j = 4;
        end

        i_col = i + (i_eval1-2)*n_vert;
        j_col = j + (i_eval1-2)*n_vert;

        k_col = i + (i_eval2-2)*n_vert;
        l_col = j + (i_eval2-2)*n_vert;

        vBase  = aInvJac(:,k_col).*( aInvJac(:,i_col)*d2Ndxii2    + aInvJac(:,j_col)*d2Ndxiidxij )
                 aInvJac(:,l_col).*( aInvJac(:,i_col)*d2Ndxijdxii + aInvJac(:,j_col)*d2Ndxij2
);
      end

  else

    vBase = 0;

  end
```

11.63 ellib/sf_simp_P2bub

SF_SIMP_P2BUB Quadratic Lagrange shape function for simplices with bubble (P2+).
[VBASE, NLDOF, XLDOF, SFUN] =

SF_SIMP_P2BUB(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming quadratic P2 Lagrange shape functions on simplices an additional with bubble function. XI Barycentric coordinates.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-4            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim+1]             Local coordinates of evaluation point
aInvJac     [n,n_sdim+1*n_sdim]    Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                            .
Output      Value/[Size]           Description
-----------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_simp_P2

Code listing

```
 [~,nLDof,xLDof] = sf_simp_P2( 0, n_sdim, n_vert );

 nLDof(4) = 1;
 switch n_sdim
   case 2
     xLDof = [ xLDof [1;1;1]/3 ];
   case 3
     xLDof = [ xLDof [1;1;1;1]/4 ];
 end
 sfun = 'sf_simp_P2bub';


% Evaluation type flag.
 vBase = 0;
```

```
if( nargin>=4 && i_dof==sum(nLDof) && n_sdim>1 )

  if( i_eval==1 )    % Evaluation of function values.

    switch n_sdim

      case 2

        vBase = 27*xi(1)*xi(2)*xi(3);

      case 3

        vBase = 256*xi(1)*xi(2)*xi(3)*xi(4);

    end

  elseif( i_eval>=2 && i_eval<=n_sdim+1 )   % Evaluation of first derivatives.

    switch n_sdim

      case 2

        dNdxi1 = 27*xi(2)*xi(3);
        dNdxi2 = 27*xi(1)*xi(3);
        dNdxi3 = 27*xi(1)*xi(2);

        if( i_eval==2 )
          vBase = aInvJac(:,1).*dNdxi1 + aInvJac(:,2).*dNdxi2 + aInvJac(:,3).*dNdxi3;
        else
          vBase = aInvJac(:,4).*dNdxi1 + aInvJac(:,5).*dNdxi2 + aInvJac(:,6).*dNdxi3;
        end

      case 3

        dNdxi1 = 256*xi(2)*xi(3)*xi(4);
        dNdxi2 = 256*xi(1)*xi(3)*xi(4);
        dNdxi3 = 256*xi(1)*xi(2)*xi(4);
        dNdxi4 = 256*xi(1)*xi(2)*xi(3);

        if( i_eval==2 )
          vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:, 2)*dNdxi2 + aInvJac(:, 3)*dNdxi3 + aInvJac(:,
        elseif( i_eval==3 )
          vBase = aInvJac(:,5)*dNdxi1 + aInvJac(:, 6)*dNdxi2 + aInvJac(:, 7)*dNdxi3 + aInvJac(:,
        else
          vBase = aInvJac(:,9)*dNdxi1 + aInvJac(:,10)*dNdxi2 + aInvJac(:,11)*dNdxi3 + aInvJac(:,1
        end

    end

  elseif( any(i_eval==[22 23 24 32 33 34 42 43 44]) )   % Evaluation of second derivatives.

    switch n_sdim
```

```
        case 2

          d2Ndxi1dxi1 = 0;
          d2Ndxi1dxi2 = 27*xi(3);
          d2Ndxi1dxi3 = 27*xi(2);
          d2Ndxi2dxi1 = 27*xi(1);
          d2Ndxi2dxi2 = 0;
          d2Ndxi2dxi3 = 27*xi(3);
          d2Ndxi3dxi1 = 27*xi(1);
          d2Ndxi3dxi2 = 27*xi(2);
          d2Ndxi3dxi3 = 0;

          if( i_eval==22 )
            vBase = aInvJac(:,1).*( aInvJac(:,1).*d2Ndxi1dxi1 + aInvJac(:,2).*d2Ndxi2dxi1 + aInvJac
                    aInvJac(:,2).*( aInvJac(:,1).*d2Ndxi1dxi2 + aInvJac(:,2).*d2Ndxi2dxi2 + aInvJac
                    aInvJac(:,3).*( aInvJac(:,1).*d2Ndxi1dxi3 + aInvJac(:,2).*d2Ndxi2dxi3 + aInvJac
          elseif( i_eval==33 )
            vBase = aInvJac(:,4).*( aInvJac(:,4).*d2Ndxi1dxi1 + aInvJac(:,5).*d2Ndxi2dxi1 + aInvJac
                    aInvJac(:,5).*( aInvJac(:,4).*d2Ndxi1dxi2 + aInvJac(:,5).*d2Ndxi2dxi2 + aInvJac
                    aInvJac(:,6).*( aInvJac(:,4).*d2Ndxi1dxi3 + aInvJac(:,5).*d2Ndxi2dxi3 + aInvJac
          else
            vBase = aInvJac(:,4).*( aInvJac(:,1).*d2Ndxi1dxi1 + aInvJac(:,2).*d2Ndxi2dxi1 + aInvJac
                    aInvJac(:,5).*( aInvJac(:,1).*d2Ndxi1dxi2 + aInvJac(:,2).*d2Ndxi2dxi2 + aInvJac
                    aInvJac(:,6).*( aInvJac(:,1).*d2Ndxi1dxi3 + aInvJac(:,2).*d2Ndxi2dxi3 + aInvJac
          end

        case 3

          d2Ndxi1dxi1 = 0;
          d2Ndxi1dxi2 = 256*xi(3)*xi(4);
          d2Ndxi1dxi3 = 256*xi(2)*xi(4);
          d2Ndxi1dxi4 = 256*xi(2)*xi(3);
          d2Ndxi2dxi1 = 256*xi(3)*xi(4);
          d2Ndxi2dxi2 = 0;
          d2Ndxi2dxi3 = 256*xi(1)*xi(4);
          d2Ndxi2dxi4 = 256*xi(1)*xi(3);
          d2Ndxi3dxi1 = 256*xi(2)*xi(4);
          d2Ndxi3dxi2 = 256*xi(1)*xi(4);
          d2Ndxi3dxi3 = 0;
          d2Ndxi3dxi4 = 256*xi(1)*xi(2);
          d2Ndxi4dxi1 = 256*xi(2)*xi(3);
          d2Ndxi4dxi2 = 256*xi(1)*xi(3);
          d2Ndxi4dxi3 = 256*xi(1)*xi(2);
          d2Ndxi4dxi4 = 0;

          switch( i_eval )
            case 22
              vBase = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJa
                      aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJa
                      aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJa
                      aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJa
```

```
            case 33
              vBase = aInvJac(:,5).*( aInvJac(:,5)*d2Ndxi1dxi1 + aInvJac(:, 6)*d2Ndxi2dxi1 + aInvJa
                      aInvJac(:,6).*( aInvJac(:,5)*d2Ndxi1dxi2 + aInvJac(:, 6)*d2Ndxi2dxi2 + aInvJa
                      aInvJac(:,7).*( aInvJac(:,5)*d2Ndxi1dxi3 + aInvJac(:, 6)*d2Ndxi2dxi3 + aInvJa
                      aInvJac(:,8).*( aInvJac(:,5)*d2Ndxi1dxi4 + aInvJac(:, 6)*d2Ndxi2dxi4 + aInvJa

            case 44
              vBase = aInvJac(:, 9).*( aInvJac(:,9)*d2Ndxi1dxi1 + aInvJac(:,10)*d2Ndxi2dxi1 + aInvJ
                      aInvJac(:,10).*( aInvJac(:,9)*d2Ndxi1dxi2 + aInvJac(:,10)*d2Ndxi2dxi2 + aInvJ
                      aInvJac(:,11).*( aInvJac(:,9)*d2Ndxi1dxi3 + aInvJac(:,10)*d2Ndxi2dxi3 + aInvJ
                      aInvJac(:,12).*( aInvJac(:,9)*d2Ndxi1dxi4 + aInvJac(:,10)*d2Ndxi2dxi4 + aInvJ

            case {23,32}
              vBase = aInvJac(:,5).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJa
                      aInvJac(:,6).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJa
                      aInvJac(:,7).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJa
                      aInvJac(:,8).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJa

            case {24,42}
              vBase = aInvJac(:, 9).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJ
                      aInvJac(:,10).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJ
                      aInvJac(:,11).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJ
                      aInvJac(:,12).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJ

            case {34,43}
              vBase = aInvJac(:, 9).*( aInvJac(:,5)*d2Ndxi1dxi1 + aInvJac(:, 6)*d2Ndxi2dxi1 + aInvJ
                      aInvJac(:,10).*( aInvJac(:,5)*d2Ndxi1dxi2 + aInvJac(:, 6)*d2Ndxi2dxi2 + aInvJ
                      aInvJac(:,11).*( aInvJac(:,5)*d2Ndxi1dxi3 + aInvJac(:, 6)*d2Ndxi2dxi3 + aInvJ
                      aInvJac(:,12).*( aInvJac(:,5)*d2Ndxi1dxi4 + aInvJac(:, 6)*d2Ndxi2dxi4 + aInvJ
          end

      end

    end

  elseif( nargin>=4 )

    vBase = sf_simp_P2( i_eval, n_sdim, n_vert, i_dof, xi, aInvJac, vBase );

  end
```

11.64 ellib/sf_tet_P3

SF_TET_P3 Third order Lagrange shape functions for tetrahedra (P3).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_TET_P3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming third order P3 Lagrange shape functions on 3D tetrahedral elements with values defined in the

nodes, edges, and faces. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 3              Number of space dimensions
n_vert      scalar: 4              Number of vertices per cell
i_dof       scalar: 1-16           Local basis function to evaluate
xi          array [4,1]            Local coordinates of evaluation point
aInvJac     [n,12]                 Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [4,n_ldof]             Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sflag3

Code listing

```
 nLDof = [4 12 4 0];
 xLDof = [1 0 0 0  2/3   0 1/3 2/3   0   0  1/3   0 2/3 1/3   0   0  1/3 1/3   0 1/3;
          0 1 0 0  1/3 2/3   0   0 2/3   0  2/3 1/3   0   0 1/3   0  1/3 1/3 1/3
0;
          0 0 1 0    0 1/3 2/3   0   0 2/3    0 2/3 1/3   0   0 1/3  1/3   0 1/3 1/3;
          0 0 0 1    0   0   0 1/3 1/3 1/3    0   0   0 2/3 2/3 2/3    0 1/3 1/3 1/3];
 sfun  = 'sf_tet_P3';


 switch i_eval

   case 1
     switch i_dof
       case 1
         vBase = (xi(1)*(3*xi(1) - 1)*(3*xi(1) - 2))/2;
       case 2
         vBase = (xi(2)*(3*xi(2) - 1)*(3*xi(2) - 2))/2;
```

```
        case 3
          vBase = (xi(3)*(3*xi(3) - 1)*(3*xi(3) - 2))/2;
        case 4
          vBase = (xi(4)*(3*xi(4) - 1)*(3*xi(4) - 2))/2;
        case 5
          vBase = (9*xi(1)*xi(2)*(3*xi(1) - 1))/2;
        case 6
          vBase = (9*xi(2)*xi(3)*(3*xi(2) - 1))/2;
        case 7
          vBase = (9*xi(1)*xi(3)*(3*xi(3) - 1))/2;
        case 8
          vBase = (9*xi(1)*xi(4)*(3*xi(1) - 1))/2;
        case 9
          vBase = (9*xi(2)*xi(4)*(3*xi(2) - 1))/2;
        case 10
          vBase = (9*xi(3)*xi(4)*(3*xi(3) - 1))/2;
        case 11
          vBase = (9*xi(1)*xi(2)*(3*xi(2) - 1))/2;
        case 12
          vBase = (9*xi(2)*xi(3)*(3*xi(3) - 1))/2;
        case 13
          vBase = (9*xi(1)*xi(3)*(3*xi(1) - 1))/2;
        case 14
          vBase = (9*xi(1)*xi(4)*(3*xi(4) - 1))/2;
        case 15
          vBase = (9*xi(2)*xi(4)*(3*xi(4) - 1))/2;
        case 16
          vBase = (9*xi(3)*xi(4)*(3*xi(4) - 1))/2;
        case 17
          vBase = 27*xi(1)*xi(2)*xi(3);
        case 18
          vBase = 27*xi(1)*xi(2)*xi(4);
        case 19
          vBase = 27*xi(2)*xi(3)*xi(4);
        case 20
          vBase = 27*xi(1)*xi(3)*xi(4);
      end

    case {2,3,4}

      switch i_dof
        case 1
          dNdxi1 = (27*xi(1)^2)/2 - 9*xi(1) + 1;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 2
          dNdxi1 = 0;
          dNdxi2 = (27*xi(2)^2)/2 - 9*xi(2) + 1;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 3
```

```
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = (27*xi(3)^2)/2 - 9*xi(3) + 1;
          dNdxi4 = 0;
        case 4
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = (27*xi(4)^2)/2 - 9*xi(4) + 1;
        case 5
          dNdxi1 = (9*xi(2)*(6*xi(1) - 1))/2;
          dNdxi2 = (9*xi(1)*(3*xi(1) - 1))/2;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 6
          dNdxi1 = 0;
          dNdxi2 = (9*xi(3)*(6*xi(2) - 1))/2;
          dNdxi3 = (9*xi(2)*(3*xi(2) - 1))/2;
          dNdxi4 = 0;
        case 7
          dNdxi1 = (9*xi(3)*(3*xi(3) - 1))/2;
          dNdxi2 = 0;
          dNdxi3 = (9*xi(1)*(6*xi(3) - 1))/2;
          dNdxi4 = 0;
        case 8
          dNdxi1 = (9*xi(4)*(6*xi(1) - 1))/2;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = (9*xi(1)*(3*xi(1) - 1))/2;
        case 9
          dNdxi1 = 0;
          dNdxi2 = (9*xi(4)*(6*xi(2) - 1))/2;
          dNdxi3 = 0;
          dNdxi4 = (9*xi(2)*(3*xi(2) - 1))/2;
        case 10
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = (9*xi(4)*(6*xi(3) - 1))/2;
          dNdxi4 = (9*xi(3)*(3*xi(3) - 1))/2;
        case 11
          dNdxi1 = (9*xi(2)*(3*xi(2) - 1))/2;
          dNdxi2 = (9*xi(1)*(6*xi(2) - 1))/2;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 12
          dNdxi1 = 0;
          dNdxi2 = (9*xi(3)*(3*xi(3) - 1))/2;
          dNdxi3 = (9*xi(2)*(6*xi(3) - 1))/2;
          dNdxi4 = 0;
        case 13
          dNdxi1 = (9*xi(3)*(6*xi(1) - 1))/2;
          dNdxi2 = 0;
```

```
            dNdxi3 = (9*xi(1)*(3*xi(1) - 1))/2;
            dNdxi4 = 0;
          case 14
            dNdxi1 = (9*xi(4)*(3*xi(4) - 1))/2;
            dNdxi2 = 0;
            dNdxi3 = 0;
            dNdxi4 = (9*xi(1)*(6*xi(4) - 1))/2;
          case 15
            dNdxi1 = 0;
            dNdxi2 = (9*xi(4)*(3*xi(4) - 1))/2;
            dNdxi3 = 0;
            dNdxi4 = (9*xi(2)*(6*xi(4) - 1))/2;
          case 16
            dNdxi1 = 0;
            dNdxi2 = 0;
            dNdxi3 = (9*xi(4)*(3*xi(4) - 1))/2;
            dNdxi4 = (9*xi(3)*(6*xi(4) - 1))/2;
          case 17
            dNdxi1 = 27*xi(2)*xi(3);
            dNdxi2 = 27*xi(1)*xi(3);
            dNdxi3 = 27*xi(1)*xi(2);
            dNdxi4 = 0;
          case 18
            dNdxi1 = 27*xi(2)*xi(4);
            dNdxi2 = 27*xi(1)*xi(4);
            dNdxi3 = 0;
            dNdxi4 = 27*xi(1)*xi(2);
          case 19
            dNdxi1 = 0;
            dNdxi2 = 27*xi(3)*xi(4);
            dNdxi3 = 27*xi(2)*xi(4);
            dNdxi4 = 27*xi(2)*xi(3);
          case 20
            dNdxi1 = 27*xi(3)*xi(4);
            dNdxi2 = 0;
            dNdxi3 = 27*xi(1)*xi(4);
            dNdxi4 = 27*xi(1)*xi(3);
        end

        if( i_eval==2 )

          vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3 + aInvJac(:,4)*dNdx

        elseif( i_eval==3 )

          vBase = aInvJac(:,5)*dNdxi1 + aInvJac(:,6)*dNdxi2 + aInvJac(:,7)*dNdxi3 + aInvJac(:,8)*dNdx

        else

          vBase = aInvJac(:,9)*dNdxi1 + aInvJac(:,10)*dNdxi2 + aInvJac(:,11)*dNdxi3 + aInvJac(:,12)*d

        end
```

```
  case {22,23,24,32,33,34,42,43,44}   % Evaluation of second derivatives.

    switch i_dof

      case 1
        d2Ndxi1dxi1 = 27*xi(1) - 9;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

      case 2
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 27*xi(2) - 9;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

      case 3
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
```

```
          d2Ndxi3dxi3 = 27*xi(3) - 9;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 4
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 27*xi(4) - 9;

        case 5
          d2Ndxi1dxi1 = 27*xi(2);
          d2Ndxi2dxi1 = 27*xi(1) - 9/2;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 27*xi(1) - 9/2;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 6
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 27*xi(3);
          d2Ndxi3dxi2 = 27*xi(2) - 9/2;
          d2Ndxi4dxi2 = 0;
```

```
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 27*xi(2) - 9/2;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 7
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 27*xi(3) - 9/2;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 27*xi(3) - 9/2;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 27*xi(1);
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 8
          d2Ndxi1dxi1 = 27*xi(4);
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 27*xi(1) - 9/2;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 27*xi(1) - 9/2;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 9
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 27*xi(4);
```

```
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 27*xi(2) - 9/2;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 27*xi(2) - 9/2;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 10
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 27*xi(4);
          d2Ndxi4dxi3 = 27*xi(3) - 9/2;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 27*xi(3) - 9/2;
          d2Ndxi4dxi4 = 0;

        case 11
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 27*xi(2) - 9/2;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 27*xi(2) - 9/2;
          d2Ndxi2dxi2 = 27*xi(1);
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 12
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
```

```
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 27*xi(3) - 9/2;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 27*xi(3) - 9/2;
          d2Ndxi3dxi3 = 27*xi(2);
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 13
          d2Ndxi1dxi1 = 27*xi(3);
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 27*xi(1) - 9/2;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 27*xi(1) - 9/2;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 14
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 27*xi(4) - 9/2;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 27*xi(4) - 9/2;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 27*xi(1);

        case 15
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
```

```
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 27*xi(4) - 9/2;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 27*xi(4) - 9/2;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 27*xi(2);

        case 16
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 27*xi(4) - 9/2;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 27*xi(4) - 9/2;
          d2Ndxi4dxi4 = 27*xi(3);

        case 17
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 27*xi(3);
          d2Ndxi3dxi1 = 27*xi(2);
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 27*xi(3);
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 27*xi(1);
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 27*xi(2);
          d2Ndxi2dxi3 = 27*xi(1);
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 18
```

```
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 27*xi(4);
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 27*xi(2);
            d2Ndxi1dxi2 = 27*xi(4);
            d2Ndxi2dxi2 = 0;
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 27*xi(1);
            d2Ndxi1dxi3 = 0;
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 0;
            d2Ndxi1dxi4 = 27*xi(2);
            d2Ndxi2dxi4 = 27*xi(1);
            d2Ndxi3dxi4 = 0;
            d2Ndxi4dxi4 = 0;

          case 19
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 0;
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 0;
            d2Ndxi1dxi2 = 0;
            d2Ndxi2dxi2 = 0;
            d2Ndxi3dxi2 = 27*xi(4);
            d2Ndxi4dxi2 = 27*xi(3);
            d2Ndxi1dxi3 = 0;
            d2Ndxi2dxi3 = 27*xi(4);
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 27*xi(2);
            d2Ndxi1dxi4 = 0;
            d2Ndxi2dxi4 = 27*xi(3);
            d2Ndxi3dxi4 = 27*xi(2);
            d2Ndxi4dxi4 = 0;

          case 20
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 0;
            d2Ndxi3dxi1 = 27*xi(4);
            d2Ndxi4dxi1 = 27*xi(3);
            d2Ndxi1dxi2 = 0;
            d2Ndxi2dxi2 = 0;
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 0;
            d2Ndxi1dxi3 = 27*xi(4);
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 27*xi(1);
            d2Ndxi1dxi4 = 27*xi(3);
            d2Ndxi2dxi4 = 0;
            d2Ndxi3dxi4 = 27*xi(1);
            d2Ndxi4dxi4 = 0;
```

```
      end

      switch( i_eval )
        case 22
          vBase = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:,
                  aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:,
                  aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:,
                  aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:,
        case 33
          vBase = aInvJac(:,5).*( aInvJac(:,5)*d2Ndxi1dxi1 + aInvJac(:, 6)*d2Ndxi2dxi1 + aInvJac(:,
                  aInvJac(:,6).*( aInvJac(:,5)*d2Ndxi1dxi2 + aInvJac(:, 6)*d2Ndxi2dxi2 + aInvJac(:,
                  aInvJac(:,7).*( aInvJac(:,5)*d2Ndxi1dxi3 + aInvJac(:, 6)*d2Ndxi2dxi3 + aInvJac(:,
                  aInvJac(:,8).*( aInvJac(:,5)*d2Ndxi1dxi4 + aInvJac(:, 6)*d2Ndxi2dxi4 + aInvJac(:,

        case 44
          vBase = aInvJac(:, 9).*( aInvJac(:,9)*d2Ndxi1dxi1 + aInvJac(:,10)*d2Ndxi2dxi1 + aInvJac(:
                  aInvJac(:,10).*( aInvJac(:,9)*d2Ndxi1dxi2 + aInvJac(:,10)*d2Ndxi2dxi2 + aInvJac(:
                  aInvJac(:,11).*( aInvJac(:,9)*d2Ndxi1dxi3 + aInvJac(:,10)*d2Ndxi2dxi3 + aInvJac(:
                  aInvJac(:,12).*( aInvJac(:,9)*d2Ndxi1dxi4 + aInvJac(:,10)*d2Ndxi2dxi4 + aInvJac(:

        case {23,32}
          vBase = aInvJac(:,5).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:,
                  aInvJac(:,6).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:,
                  aInvJac(:,7).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:,
                  aInvJac(:,8).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:,

        case {24,42}
          vBase = aInvJac(:, 9).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:
                  aInvJac(:,10).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:
                  aInvJac(:,11).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:
                  aInvJac(:,12).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:

        case {34,43}
          vBase = aInvJac(:, 9).*( aInvJac(:,5)*d2Ndxi1dxi1 + aInvJac(:, 6)*d2Ndxi2dxi1 + aInvJac(:
                  aInvJac(:,10).*( aInvJac(:,5)*d2Ndxi1dxi2 + aInvJac(:, 6)*d2Ndxi2dxi2 + aInvJac(:
                  aInvJac(:,11).*( aInvJac(:,5)*d2Ndxi1dxi3 + aInvJac(:, 6)*d2Ndxi2dxi3 + aInvJac(:
                  aInvJac(:,12).*( aInvJac(:,5)*d2Ndxi1dxi4 + aInvJac(:, 6)*d2Ndxi2dxi4 + aInvJac(:
      end

    otherwise

      vBase = 0;

  end
```

11.65 ellib/sf_tet_P4

SF_TET_P4 Fourth order Lagrange shape functions for tetrahedra (P4).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_TET_P4(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming fourth order P4 Lagrange shape functions on 3D tetrahedral elements with values defined in the nodes, edges, faces, and cell center. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 3              Number of space dimensions
n_vert      scalar: 4              Number of vertices per cell
i_dof       scalar: 1-35           Local basis function to evaluate
xi          array [4,1]            Local coordinates of evaluation point
aInvJac     [n,12]                 Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [4,n_ldof]             Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_tet_P1

Code listing

```
 nLDof = [4 18 12 1];
 xLDof = [1 0 0 0 3/4   0 1/4 3/4   0   0 1/2   0 1/2 1/2   0   0 1/4   0 3/4 1/4
0   0 1/2 1/2   0 1/4 1/4 1/4   0 1/2 1/4 1/4   0 1/4 1/4;
          0 1 0 0 1/4 3/4   0   0 3/4   0 1/2 1/2   0   0 1/2   0 3/4 1/4   0   0 1/4
0 1/4 1/4 1/2   0 1/2 1/2 1/4   0 1/4 1/4 1/4   0 1/4;
          0 0 1 0   0 1/4 3/4   0   0 3/4   0 1/2 1/2   0   0 1/2   0 3/4 1/4   0
0 1/4 1/4   0 1/4 1/2 1/4   0 1/2 1/4 1/2   0 1/4 1/4 1/4;
          0 0 0 1   0   0   0 1/4 1/4 1/4   0   0   0 1/2 1/2 1/2   0   0   0 3/4 3/4 3/4
0 1/4 1/4 1/4   0 1/4 1/4 1/4   0 1/2 1/2 1/2 1/4];
 sfun  = 'sf_tet_P4';


 switch i_eval
```

```
case 1

  switch i_dof
    case 1
      vBase = (xi(1)*(2*xi(1) - 1)*(4*xi(1) - 1)*(4*xi(1) - 3))/3;
    case 2
      vBase = (xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1)*(4*xi(2) - 3))/3;
    case 3
      vBase = (xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1)*(4*xi(3) - 3))/3;
    case 4
      vBase = (xi(4)*(2*xi(4) - 1)*(4*xi(4) - 1)*(4*xi(4) - 3))/3;
    case 5
      vBase = (16*xi(1)*xi(2)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
    case 6
      vBase = (16*xi(2)*xi(3)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
    case 7
      vBase = (16*xi(1)*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
    case 8
      vBase = (16*xi(1)*xi(4)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
    case 9
      vBase = (16*xi(2)*xi(4)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
    case 10
      vBase = (16*xi(3)*xi(4)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
    case 11
      vBase = 4*xi(1)*xi(2)*(4*xi(1) - 1)*(4*xi(2) - 1);
    case 12
      vBase = 4*xi(2)*xi(3)*(4*xi(2) - 1)*(4*xi(3) - 1);
    case 13
      vBase = 4*xi(1)*xi(3)*(4*xi(1) - 1)*(4*xi(3) - 1);
    case 14
      vBase = 4*xi(1)*xi(4)*(4*xi(1) - 1)*(4*xi(4) - 1);
    case 15
      vBase = 4*xi(2)*xi(4)*(4*xi(2) - 1)*(4*xi(4) - 1);
    case 16
      vBase = 4*xi(3)*xi(4)*(4*xi(3) - 1)*(4*xi(4) - 1);
    case 17
      vBase = (16*xi(1)*xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
    case 18
      vBase = (16*xi(2)*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
    case 19
      vBase = (16*xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
    case 20
      vBase = (16*xi(1)*xi(4)*(2*xi(4) - 1)*(4*xi(4) - 1))/3;
    case 21
      vBase = (16*xi(2)*xi(4)*(2*xi(4) - 1)*(4*xi(4) - 1))/3;
    case 22
      vBase = (16*xi(3)*xi(4)*(2*xi(4) - 1)*(4*xi(4) - 1))/3;
    case 23
      vBase = 32*xi(1)*xi(2)*xi(3)*(4*xi(1) - 1);
    case 24
      vBase = 32*xi(1)*xi(2)*xi(4)*(4*xi(1) - 1);
```

```
        case 25
          vBase = 32*xi(2)*xi(3)*xi(4)*(4*xi(2) - 1);
        case 26
          vBase = 32*xi(1)*xi(3)*xi(4)*(4*xi(3) - 1);
        case 27
          vBase = 32*xi(1)*xi(2)*xi(3)*(4*xi(2) - 1);
        case 28
          vBase = 32*xi(1)*xi(2)*xi(4)*(4*xi(2) - 1);
        case 29
          vBase = 32*xi(2)*xi(3)*xi(4)*(4*xi(3) - 1);
        case 30
          vBase = 32*xi(1)*xi(3)*xi(4)*(4*xi(1) - 1);
        case 31
          vBase = 32*xi(1)*xi(2)*xi(3)*(4*xi(3) - 1);
        case 32
          vBase = 32*xi(1)*xi(2)*xi(4)*(4*xi(4) - 1);
        case 33
          vBase = 32*xi(2)*xi(3)*xi(4)*(4*xi(4) - 1);
        case 34
          vBase = 32*xi(1)*xi(3)*xi(4)*(4*xi(4) - 1);
        case 35
          vBase = 256*xi(1)*xi(2)*xi(3)*xi(4);
      end

    case {2,3,4}

      switch i_dof

        case 1
          dNdxi1 = ((8*xi(1) - 3)*(16*xi(1)^2 - 12*xi(1) + 1))/3;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 2
          dNdxi1 = 0;
          dNdxi2 = ((8*xi(2) - 3)*(16*xi(2)^2 - 12*xi(2) + 1))/3;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 3
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = ((8*xi(3) - 3)*(16*xi(3)^2 - 12*xi(3) + 1))/3;
          dNdxi4 = 0;
        case 4
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = ((8*xi(4) - 3)*(16*xi(4)^2 - 12*xi(4) + 1))/3;
        case 5
          dNdxi1 = (16*xi(2)*(24*xi(1)^2 - 12*xi(1) + 1))/3;
          dNdxi2 = (16*xi(1)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
          dNdxi3 = 0;
```

```
          dNdxi4 = 0;
        case 6
          dNdxi1 = 0;
          dNdxi2 = (16*xi(3)*(24*xi(2)^2 - 12*xi(2) + 1))/3;
          dNdxi3 = (16*xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
          dNdxi4 = 0;
        case 7
          dNdxi1 = (16*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
          dNdxi2 = 0;
          dNdxi3 = (16*xi(1)*(24*xi(3)^2 - 12*xi(3) + 1))/3;
          dNdxi4 = 0;
        case 8
          dNdxi1 = (16*xi(4)*(24*xi(1)^2 - 12*xi(1) + 1))/3;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = (16*xi(1)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
        case 9
          dNdxi1 = 0;
          dNdxi2 = (16*xi(4)*(24*xi(2)^2 - 12*xi(2) + 1))/3;
          dNdxi3 = 0;
          dNdxi4 = (16*xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
        case 10
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = (16*xi(4)*(24*xi(3)^2 - 12*xi(3) + 1))/3;
          dNdxi4 = (16*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
        case 11
          dNdxi1 = 4*xi(2)*(8*xi(1) - 1)*(4*xi(2) - 1);
          dNdxi2 = 4*xi(1)*(4*xi(1) - 1)*(8*xi(2) - 1);
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 12
          dNdxi1 = 0;
          dNdxi2 = 4*xi(3)*(8*xi(2) - 1)*(4*xi(3) - 1);
          dNdxi3 = 4*xi(2)*(4*xi(2) - 1)*(8*xi(3) - 1);
          dNdxi4 = 0;
        case 13
          dNdxi1 = 4*xi(3)*(8*xi(1) - 1)*(4*xi(3) - 1);
          dNdxi2 = 0;
          dNdxi3 = 4*xi(1)*(4*xi(1) - 1)*(8*xi(3) - 1);
          dNdxi4 = 0;
        case 14
          dNdxi1 = 4*xi(4)*(8*xi(1) - 1)*(4*xi(4) - 1);
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = 4*xi(1)*(4*xi(1) - 1)*(8*xi(4) - 1);
        case 15
          dNdxi1 = 0;
          dNdxi2 = 4*xi(4)*(8*xi(2) - 1)*(4*xi(4) - 1);
          dNdxi3 = 0;
          dNdxi4 = 4*xi(2)*(4*xi(2) - 1)*(8*xi(4) - 1);
        case 16
```

```
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = 4*xi(4)*(8*xi(3) - 1)*(4*xi(4) - 1);
          dNdxi4 = 4*xi(3)*(4*xi(3) - 1)*(8*xi(4) - 1);
        case 17
          dNdxi1 = (16*xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
          dNdxi2 = (16*xi(1)*(24*xi(2)^2 - 12*xi(2) + 1))/3;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 18
          dNdxi1 = 0;
          dNdxi2 = (16*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
          dNdxi3 = (16*xi(2)*(24*xi(3)^2 - 12*xi(3) + 1))/3;
          dNdxi4 = 0;
        case 19
          dNdxi1 = (16*xi(3)*(24*xi(1)^2 - 12*xi(1) + 1))/3;
          dNdxi2 = 0;
          dNdxi3 = (16*xi(1)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
          dNdxi4 = 0;
        case 20
          dNdxi1 = (16*xi(4)*(2*xi(4) - 1)*(4*xi(4) - 1))/3;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = (16*xi(1)*(24*xi(4)^2 - 12*xi(4) + 1))/3;
        case 21
          dNdxi1 = 0;
          dNdxi2 = (16*xi(4)*(2*xi(4) - 1)*(4*xi(4) - 1))/3;
          dNdxi3 = 0;
          dNdxi4 = (16*xi(2)*(24*xi(4)^2 - 12*xi(4) + 1))/3;
        case 22
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = (16*xi(4)*(2*xi(4) - 1)*(4*xi(4) - 1))/3;
          dNdxi4 = (16*xi(3)*(24*xi(4)^2 - 12*xi(4) + 1))/3;
        case 23
          dNdxi1 = 32*xi(2)*xi(3)*(8*xi(1) - 1);
          dNdxi2 = 32*xi(1)*xi(3)*(4*xi(1) - 1);
          dNdxi3 = 32*xi(1)*xi(2)*(4*xi(1) - 1);
          dNdxi4 = 0;
        case 24
          dNdxi1 = 32*xi(2)*xi(4)*(8*xi(1) - 1);
          dNdxi2 = 32*xi(1)*xi(4)*(4*xi(1) - 1);
          dNdxi3 = 0;
          dNdxi4 = 32*xi(1)*xi(2)*(4*xi(1) - 1);
        case 25
          dNdxi1 = 0;
          dNdxi2 = 32*xi(3)*xi(4)*(8*xi(2) - 1);
          dNdxi3 = 32*xi(2)*xi(4)*(4*xi(2) - 1);
          dNdxi4 = 32*xi(2)*xi(3)*(4*xi(2) - 1);
        case 26
          dNdxi1 = 32*xi(3)*xi(4)*(4*xi(3) - 1);
          dNdxi2 = 0;
```

```
          dNdxi3 = 32*xi(1)*xi(4)*(8*xi(3) - 1);
          dNdxi4 = 32*xi(1)*xi(3)*(4*xi(3) - 1);
        case 27
          dNdxi1 = 32*xi(2)*xi(3)*(4*xi(2) - 1);
          dNdxi2 = 32*xi(1)*xi(3)*(8*xi(2) - 1);
          dNdxi3 = 32*xi(1)*xi(2)*(4*xi(2) - 1);
          dNdxi4 = 0;
        case 28
          dNdxi1 = 32*xi(2)*xi(4)*(4*xi(2) - 1);
          dNdxi2 = 32*xi(1)*xi(4)*(8*xi(2) - 1);
          dNdxi3 = 0;
          dNdxi4 = 32*xi(1)*xi(2)*(4*xi(2) - 1);
        case 29
          dNdxi1 = 0;
          dNdxi2 = 32*xi(3)*xi(4)*(4*xi(3) - 1);
          dNdxi3 = 32*xi(2)*xi(4)*(8*xi(3) - 1);
          dNdxi4 = 32*xi(2)*xi(3)*(4*xi(3) - 1);
        case 30
          dNdxi1 = 32*xi(3)*xi(4)*(8*xi(1) - 1);
          dNdxi2 = 0;
          dNdxi3 = 32*xi(1)*xi(4)*(4*xi(1) - 1);
          dNdxi4 = 32*xi(1)*xi(3)*(4*xi(1) - 1);
        case 31
          dNdxi1 = 32*xi(2)*xi(3)*(4*xi(3) - 1);
          dNdxi2 = 32*xi(1)*xi(3)*(4*xi(3) - 1);
          dNdxi3 = 32*xi(1)*xi(2)*(8*xi(3) - 1);
          dNdxi4 = 0;
        case 32
          dNdxi1 = 32*xi(2)*xi(4)*(4*xi(4) - 1);
          dNdxi2 = 32*xi(1)*xi(4)*(4*xi(4) - 1);
          dNdxi3 = 0;
          dNdxi4 = 32*xi(1)*xi(2)*(8*xi(4) - 1);
        case 33
          dNdxi1 = 0;
          dNdxi2 = 32*xi(3)*xi(4)*(4*xi(4) - 1);
          dNdxi3 = 32*xi(2)*xi(4)*(4*xi(4) - 1);
          dNdxi4 = 32*xi(2)*xi(3)*(8*xi(4) - 1);
        case 34
          dNdxi1 = 32*xi(3)*xi(4)*(4*xi(4) - 1);
          dNdxi2 = 0;
          dNdxi3 = 32*xi(1)*xi(4)*(4*xi(4) - 1);
          dNdxi4 = 32*xi(1)*xi(3)*(8*xi(4) - 1);
        case 35
          dNdxi1 = 256*xi(2)*xi(3)*xi(4);
          dNdxi2 = 256*xi(1)*xi(3)*xi(4);
          dNdxi3 = 256*xi(1)*xi(2)*xi(4);
          dNdxi4 = 256*xi(1)*xi(2)*xi(3);
      end

      if( i_eval==2 )

        vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3 + aInvJac(:,4)*dNdx
```

```
    elseif( i_eval==3 )

      vBase = aInvJac(:,5)*dNdxi1 + aInvJac(:,6)*dNdxi2 + aInvJac(:,7)*dNdxi3 + aInvJac(:,8)*dNdx

    else

      vBase = aInvJac(:,9)*dNdxi1 + aInvJac(:,10)*dNdxi2 + aInvJac(:,11)*dNdxi3 + aInvJac(:,12)*d

    end

  case {22,23,24,32,33,34,42,43,44}   % Evaluation of second derivatives.

    switch i_dof

      case 1
        d2Ndxi1dxi1 = 128*xi(1)^2 - 96*xi(1) + 44/3;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

      case 2
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 128*xi(2)^2 - 96*xi(2) + 44/3;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

      case 3
```

```
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 0;
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 0;
            d2Ndxi1dxi2 = 0;
            d2Ndxi2dxi2 = 0;
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 0;
            d2Ndxi1dxi3 = 0;
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 128*xi(3)^2 - 96*xi(3) + 44/3;
            d2Ndxi4dxi3 = 0;
            d2Ndxi1dxi4 = 0;
            d2Ndxi2dxi4 = 0;
            d2Ndxi3dxi4 = 0;
            d2Ndxi4dxi4 = 0;

          case 4
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 0;
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 0;
            d2Ndxi1dxi2 = 0;
            d2Ndxi2dxi2 = 0;
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 0;
            d2Ndxi1dxi3 = 0;
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 0;
            d2Ndxi1dxi4 = 0;
            d2Ndxi2dxi4 = 0;
            d2Ndxi3dxi4 = 0;
            d2Ndxi4dxi4 = 128*xi(4)^2 - 96*xi(4) + 44/3;

          case 5
            d2Ndxi1dxi1 = 64*xi(2)*(4*xi(1) - 1);
            d2Ndxi2dxi1 = 128*xi(1)^2 - 64*xi(1) + 16/3;
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 0;
            d2Ndxi1dxi2 = 128*xi(1)^2 - 64*xi(1) + 16/3;
            d2Ndxi2dxi2 = 0;
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 0;
            d2Ndxi1dxi3 = 0;
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 0;
            d2Ndxi1dxi4 = 0;
            d2Ndxi2dxi4 = 0;
            d2Ndxi3dxi4 = 0;
            d2Ndxi4dxi4 = 0;
```

```
        case 6
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 64*xi(3)*(4*xi(2) - 1);
          d2Ndxi3dxi2 = 128*xi(2)^2 - 64*xi(2) + 16/3;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 128*xi(2)^2 - 64*xi(2) + 16/3;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 7
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 128*xi(3)^2 - 64*xi(3) + 16/3;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 128*xi(3)^2 - 64*xi(3) + 16/3;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 64*xi(1)*(4*xi(3) - 1);
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 8
          d2Ndxi1dxi1 = 64*xi(4)*(4*xi(1) - 1);
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 128*xi(1)^2 - 64*xi(1) + 16/3;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 128*xi(1)^2 - 64*xi(1) + 16/3;
          d2Ndxi2dxi4 = 0;
```

```
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 9
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 64*xi(4)*(4*xi(2) - 1);
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 128*xi(2)^2 - 64*xi(2) + 16/3;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 128*xi(2)^2 - 64*xi(2) + 16/3;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 10
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 64*xi(4)*(4*xi(3) - 1);
    d2Ndxi4dxi3 = 128*xi(3)^2 - 64*xi(3) + 16/3;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 128*xi(3)^2 - 64*xi(3) + 16/3;
    d2Ndxi4dxi4 = 0;

  case 11
    d2Ndxi1dxi1 = 32*xi(2)*(4*xi(2) - 1);
    d2Ndxi2dxi1 = 4*(8*xi(1) - 1)*(8*xi(2) - 1);
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 4*(8*xi(1) - 1)*(8*xi(2) - 1);
    d2Ndxi2dxi2 = 32*xi(1)*(4*xi(1) - 1);
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
```

```
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 12
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 32*xi(3)*(4*xi(3) - 1);
          d2Ndxi3dxi2 = 4*(8*xi(2) - 1)*(8*xi(3) - 1);
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 4*(8*xi(2) - 1)*(8*xi(3) - 1);
          d2Ndxi3dxi3 = 32*xi(2)*(4*xi(2) - 1);
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 13
          d2Ndxi1dxi1 = 32*xi(3)*(4*xi(3) - 1);
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 4*(8*xi(1) - 1)*(8*xi(3) - 1);
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 4*(8*xi(1) - 1)*(8*xi(3) - 1);
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 32*xi(1)*(4*xi(1) - 1);
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 14
          d2Ndxi1dxi1 = 32*xi(4)*(4*xi(4) - 1);
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 4*(8*xi(1) - 1)*(8*xi(4) - 1);
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
```

```
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 0;
            d2Ndxi1dxi4 = 4*(8*xi(1) - 1)*(8*xi(4) - 1);
            d2Ndxi2dxi4 = 0;
            d2Ndxi3dxi4 = 0;
            d2Ndxi4dxi4 = 32*xi(1)*(4*xi(1) - 1);

          case 15
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 0;
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 0;
            d2Ndxi1dxi2 = 0;
            d2Ndxi2dxi2 = 32*xi(4)*(4*xi(4) - 1);
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 4*(8*xi(2) - 1)*(8*xi(4) - 1);
            d2Ndxi1dxi3 = 0;
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 0;
            d2Ndxi1dxi4 = 0;
            d2Ndxi2dxi4 = 4*(8*xi(2) - 1)*(8*xi(4) - 1);
            d2Ndxi3dxi4 = 0;
            d2Ndxi4dxi4 = 32*xi(2)*(4*xi(2) - 1);

          case 16
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 0;
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 0;
            d2Ndxi1dxi2 = 0;
            d2Ndxi2dxi2 = 0;
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 0;
            d2Ndxi1dxi3 = 0;
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 32*xi(4)*(4*xi(4) - 1);
            d2Ndxi4dxi3 = 4*(8*xi(3) - 1)*(8*xi(4) - 1);
            d2Ndxi1dxi4 = 0;
            d2Ndxi2dxi4 = 0;
            d2Ndxi3dxi4 = 4*(8*xi(3) - 1)*(8*xi(4) - 1);
            d2Ndxi4dxi4 = 32*xi(3)*(4*xi(3) - 1);

          case 17
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 128*xi(2)^2 - 64*xi(2) + 16/3;
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 0;
            d2Ndxi1dxi2 = 128*xi(2)^2 - 64*xi(2) + 16/3;
            d2Ndxi2dxi2 = 64*xi(1)*(4*xi(2) - 1);
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 0;
```

```
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 18
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 128*xi(3)^2 - 64*xi(3) + 16/3;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 128*xi(3)^2 - 64*xi(3) + 16/3;
          d2Ndxi3dxi3 = 64*xi(2)*(4*xi(3) - 1);
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 19
          d2Ndxi1dxi1 = 64*xi(3)*(4*xi(1) - 1);
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 128*xi(1)^2 - 64*xi(1) + 16/3;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 128*xi(1)^2 - 64*xi(1) + 16/3;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 20
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 128*xi(4)^2 - 64*xi(4) + 16/3;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
```

```
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 128*xi(4)^2 - 64*xi(4) + 16/3;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 64*xi(1)*(4*xi(4) - 1);

        case 21
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 128*xi(4)^2 - 64*xi(4) + 16/3;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 128*xi(4)^2 - 64*xi(4) + 16/3;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 64*xi(2)*(4*xi(4) - 1);

        case 22
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 128*xi(4)^2 - 64*xi(4) + 16/3;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 128*xi(4)^2 - 64*xi(4) + 16/3;
          d2Ndxi4dxi4 = 64*xi(3)*(4*xi(4) - 1);

        case 23
          d2Ndxi1dxi1 = 256*xi(2)*xi(3);
          d2Ndxi2dxi1 = 32*xi(3)*(8*xi(1) - 1);
          d2Ndxi3dxi1 = 32*xi(2)*(8*xi(1) - 1);
          d2Ndxi4dxi1 = 0;
```

```
      d2Ndxi1dxi2 = 32*xi(3)*(8*xi(1) - 1);
      d2Ndxi2dxi2 = 0;
      d2Ndxi3dxi2 = 32*xi(1)*(4*xi(1) - 1);
      d2Ndxi4dxi2 = 0;
      d2Ndxi1dxi3 = 32*xi(2)*(8*xi(1) - 1);
      d2Ndxi2dxi3 = 32*xi(1)*(4*xi(1) - 1);
      d2Ndxi3dxi3 = 0;
      d2Ndxi4dxi3 = 0;
      d2Ndxi1dxi4 = 0;
      d2Ndxi2dxi4 = 0;
      d2Ndxi3dxi4 = 0;
      d2Ndxi4dxi4 = 0;

    case 24
      d2Ndxi1dxi1 = 256*xi(2)*xi(4);
      d2Ndxi2dxi1 = 32*xi(4)*(8*xi(1) - 1);
      d2Ndxi3dxi1 = 0;
      d2Ndxi4dxi1 = 32*xi(2)*(8*xi(1) - 1);
      d2Ndxi1dxi2 = 32*xi(4)*(8*xi(1) - 1);
      d2Ndxi2dxi2 = 0;
      d2Ndxi3dxi2 = 0;
      d2Ndxi4dxi2 = 32*xi(1)*(4*xi(1) - 1);
      d2Ndxi1dxi3 = 0;
      d2Ndxi2dxi3 = 0;
      d2Ndxi3dxi3 = 0;
      d2Ndxi4dxi3 = 0;
      d2Ndxi1dxi4 = 32*xi(2)*(8*xi(1) - 1);
      d2Ndxi2dxi4 = 32*xi(1)*(4*xi(1) - 1);
      d2Ndxi3dxi4 = 0;
      d2Ndxi4dxi4 = 0;

    case 25
      d2Ndxi1dxi1 = 0;
      d2Ndxi2dxi1 = 0;
      d2Ndxi3dxi1 = 0;
      d2Ndxi4dxi1 = 0;
      d2Ndxi1dxi2 = 0;
      d2Ndxi2dxi2 = 256*xi(3)*xi(4);
      d2Ndxi3dxi2 = 32*xi(4)*(8*xi(2) - 1);
      d2Ndxi4dxi2 = 32*xi(3)*(8*xi(2) - 1);
      d2Ndxi1dxi3 = 0;
      d2Ndxi2dxi3 = 32*xi(4)*(8*xi(2) - 1);
      d2Ndxi3dxi3 = 0;
      d2Ndxi4dxi3 = 32*xi(2)*(4*xi(2) - 1);
      d2Ndxi1dxi4 = 0;
      d2Ndxi2dxi4 = 32*xi(3)*(8*xi(2) - 1);
      d2Ndxi3dxi4 = 32*xi(2)*(4*xi(2) - 1);
      d2Ndxi4dxi4 = 0;

    case 26
      d2Ndxi1dxi1 = 0;
      d2Ndxi2dxi1 = 0;
```

```
            d2Ndxi3dxi1 = 32*xi(4)*(8*xi(3) - 1);
            d2Ndxi4dxi1 = 32*xi(3)*(4*xi(3) - 1);
            d2Ndxi1dxi2 = 0;
            d2Ndxi2dxi2 = 0;
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 0;
            d2Ndxi1dxi3 = 32*xi(4)*(8*xi(3) - 1);
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 256*xi(1)*xi(4);
            d2Ndxi4dxi3 = 32*xi(1)*(8*xi(3) - 1);
            d2Ndxi1dxi4 = 32*xi(3)*(4*xi(3) - 1);
            d2Ndxi2dxi4 = 0;
            d2Ndxi3dxi4 = 32*xi(1)*(8*xi(3) - 1);
            d2Ndxi4dxi4 = 0;

          case 27
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 32*xi(3)*(8*xi(2) - 1);
            d2Ndxi3dxi1 = 32*xi(2)*(4*xi(2) - 1);
            d2Ndxi4dxi1 = 0;
            d2Ndxi1dxi2 = 32*xi(3)*(8*xi(2) - 1);
            d2Ndxi2dxi2 = 256*xi(1)*xi(3);
            d2Ndxi3dxi2 = 32*xi(1)*(8*xi(2) - 1);
            d2Ndxi4dxi2 = 0;
            d2Ndxi1dxi3 = 32*xi(2)*(4*xi(2) - 1);
            d2Ndxi2dxi3 = 32*xi(1)*(8*xi(2) - 1);
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 0;
            d2Ndxi1dxi4 = 0;
            d2Ndxi2dxi4 = 0;
            d2Ndxi3dxi4 = 0;
            d2Ndxi4dxi4 = 0;

          case 28
            d2Ndxi1dxi1 = 0;
            d2Ndxi2dxi1 = 32*xi(4)*(8*xi(2) - 1);
            d2Ndxi3dxi1 = 0;
            d2Ndxi4dxi1 = 32*xi(2)*(4*xi(2) - 1);
            d2Ndxi1dxi2 = 32*xi(4)*(8*xi(2) - 1);
            d2Ndxi2dxi2 = 256*xi(1)*xi(4);
            d2Ndxi3dxi2 = 0;
            d2Ndxi4dxi2 = 32*xi(1)*(8*xi(2) - 1);
            d2Ndxi1dxi3 = 0;
            d2Ndxi2dxi3 = 0;
            d2Ndxi3dxi3 = 0;
            d2Ndxi4dxi3 = 0;
            d2Ndxi1dxi4 = 32*xi(2)*(4*xi(2) - 1);
            d2Ndxi2dxi4 = 32*xi(1)*(8*xi(2) - 1);
            d2Ndxi3dxi4 = 0;
            d2Ndxi4dxi4 = 0;

          case 29
```

```
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 32*xi(4)*(8*xi(3) - 1);
          d2Ndxi4dxi2 = 32*xi(3)*(4*xi(3) - 1);
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 32*xi(4)*(8*xi(3) - 1);
          d2Ndxi3dxi3 = 256*xi(2)*xi(4);
          d2Ndxi4dxi3 = 32*xi(2)*(8*xi(3) - 1);
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 32*xi(3)*(4*xi(3) - 1);
          d2Ndxi3dxi4 = 32*xi(2)*(8*xi(3) - 1);
          d2Ndxi4dxi4 = 0;

        case 30
          d2Ndxi1dxi1 = 256*xi(3)*xi(4);
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 32*xi(4)*(8*xi(1) - 1);
          d2Ndxi4dxi1 = 32*xi(3)*(8*xi(1) - 1);
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 32*xi(4)*(8*xi(1) - 1);
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 32*xi(1)*(4*xi(1) - 1);
          d2Ndxi1dxi4 = 32*xi(3)*(8*xi(1) - 1);
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 32*xi(1)*(4*xi(1) - 1);
          d2Ndxi4dxi4 = 0;

        case 31
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 32*xi(3)*(4*xi(3) - 1);
          d2Ndxi3dxi1 = 32*xi(2)*(8*xi(3) - 1);
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 32*xi(3)*(4*xi(3) - 1);
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 32*xi(1)*(8*xi(3) - 1);
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 32*xi(2)*(8*xi(3) - 1);
          d2Ndxi2dxi3 = 32*xi(1)*(8*xi(3) - 1);
          d2Ndxi3dxi3 = 256*xi(1)*xi(2);
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;
```

```
        case 32
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 32*xi(4)*(4*xi(4) - 1);
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 32*xi(2)*(8*xi(4) - 1);
          d2Ndxi1dxi2 = 32*xi(4)*(4*xi(4) - 1);
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 32*xi(1)*(8*xi(4) - 1);
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 32*xi(2)*(8*xi(4) - 1);
          d2Ndxi2dxi4 = 32*xi(1)*(8*xi(4) - 1);
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 256*xi(1)*xi(2);

        case 33
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 32*xi(4)*(4*xi(4) - 1);
          d2Ndxi4dxi2 = 32*xi(3)*(8*xi(4) - 1);
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 32*xi(4)*(4*xi(4) - 1);
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 32*xi(2)*(8*xi(4) - 1);
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 32*xi(3)*(8*xi(4) - 1);
          d2Ndxi3dxi4 = 32*xi(2)*(8*xi(4) - 1);
          d2Ndxi4dxi4 = 256*xi(2)*xi(3);

        case 34
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 32*xi(4)*(4*xi(4) - 1);
          d2Ndxi4dxi1 = 32*xi(3)*(8*xi(4) - 1);
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 32*xi(4)*(4*xi(4) - 1);
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 32*xi(1)*(8*xi(4) - 1);
          d2Ndxi1dxi4 = 32*xi(3)*(8*xi(4) - 1);
          d2Ndxi2dxi4 = 0;
```

```
      d2Ndxi3dxi4 = 32*xi(1)*(8*xi(4) - 1);
      d2Ndxi4dxi4 = 256*xi(1)*xi(3);

    case 35
      d2Ndxi1dxi1 = 0;
      d2Ndxi2dxi1 = 256*xi(3)*xi(4);
      d2Ndxi3dxi1 = 256*xi(2)*xi(4);
      d2Ndxi4dxi1 = 256*xi(2)*xi(3);
      d2Ndxi1dxi2 = 256*xi(3)*xi(4);
      d2Ndxi2dxi2 = 0;
      d2Ndxi3dxi2 = 256*xi(1)*xi(4);
      d2Ndxi4dxi2 = 256*xi(1)*xi(3);
      d2Ndxi1dxi3 = 256*xi(2)*xi(4);
      d2Ndxi2dxi3 = 256*xi(1)*xi(4);
      d2Ndxi3dxi3 = 0;
      d2Ndxi4dxi3 = 256*xi(1)*xi(2);
      d2Ndxi1dxi4 = 256*xi(2)*xi(3);
      d2Ndxi2dxi4 = 256*xi(1)*xi(3);
      d2Ndxi3dxi4 = 256*xi(1)*xi(2);
      d2Ndxi4dxi4 = 0;

  end

  switch( i_eval )
    case 22
      vBase = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:,
              aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:,
              aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:,
              aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:,
    case 33
      vBase = aInvJac(:,5).*( aInvJac(:,5)*d2Ndxi1dxi1 + aInvJac(:, 6)*d2Ndxi2dxi1 + aInvJac(:,
              aInvJac(:,6).*( aInvJac(:,5)*d2Ndxi1dxi2 + aInvJac(:, 6)*d2Ndxi2dxi2 + aInvJac(:,
              aInvJac(:,7).*( aInvJac(:,5)*d2Ndxi1dxi3 + aInvJac(:, 6)*d2Ndxi2dxi3 + aInvJac(:,
              aInvJac(:,8).*( aInvJac(:,5)*d2Ndxi1dxi4 + aInvJac(:, 6)*d2Ndxi2dxi4 + aInvJac(:,

    case 44
      vBase = aInvJac(:, 9).*( aInvJac(:,9)*d2Ndxi1dxi1 + aInvJac(:,10)*d2Ndxi2dxi1 + aInvJac(:
              aInvJac(:,10).*( aInvJac(:,9)*d2Ndxi1dxi2 + aInvJac(:,10)*d2Ndxi2dxi2 + aInvJac(:
              aInvJac(:,11).*( aInvJac(:,9)*d2Ndxi1dxi3 + aInvJac(:,10)*d2Ndxi2dxi3 + aInvJac(:
              aInvJac(:,12).*( aInvJac(:,9)*d2Ndxi1dxi4 + aInvJac(:,10)*d2Ndxi2dxi4 + aInvJac(:

    case {23,32}
      vBase = aInvJac(:,5).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:,
              aInvJac(:,6).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:,
              aInvJac(:,7).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:,
              aInvJac(:,8).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:,

    case {24,42}
      vBase = aInvJac(:, 9).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:
              aInvJac(:,10).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:
              aInvJac(:,11).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:
              aInvJac(:,12).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:
```

```
       case {34,43}
         vBase = aInvJac(:, 9).*( aInvJac(:,5)*d2Ndxi1dxi1 + aInvJac(:, 6)*d2Ndxi2dxi1 + aInvJac(:
                 aInvJac(:,10).*( aInvJac(:,5)*d2Ndxi1dxi2 + aInvJac(:, 6)*d2Ndxi2dxi2 + aInvJac(:
                 aInvJac(:,11).*( aInvJac(:,5)*d2Ndxi1dxi3 + aInvJac(:, 6)*d2Ndxi2dxi3 + aInvJac(:
                 aInvJac(:,12).*( aInvJac(:,5)*d2Ndxi1dxi4 + aInvJac(:, 6)*d2Ndxi2dxi4 + aInvJac(:
     end

   otherwise
     vBase = 0;

 end
```

11.66 ellib/sf_tet_P5

SF_TET_P5 Fifth order Lagrange shape functions for tetrahedra (P5).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_TET_P5(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming fifth order P5 Lagrange shape functions on 3D tetrahedral elements with values defined in the nodes, edges, faces, and cell center. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 3              Number of space dimensions
n_vert      scalar: 4              Number of vertices per cell
i_dof       scalar: 1-56           Local basis function to evaluate
xi          array [4,1]            Local coordinates of evaluation point
aInvJac     [n,12]                 Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                            .
Output      Value/[Size]           Description
-----------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [4,n_ldof]             Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_tet_P1

Code listing

```
 nLDof = [4 24 24 4];
 xLDof = [1 0 0 0 4/5   0 1/5 4/5   0   0 3/5   0 2/5 3/5   0   0 2/5   0 3/5 2/5
0   0 1/5   0 4/5 1/5   0   0 3/5 3/5   0 1/5 2/5 2/5   0 2/5 1/5 1/5   0 3/5 1/5 1/5
0 2/5 1/5 1/5   0 1/5 2/5 2/5   0 1/5 2/5 1/5 1/5 1/5;
          0 1 0 0 1/5 4/5   0   0 4/5   0 2/5 3/5   0   0 3/5   0 3/5 2/5   0   0 2/5
0 4/5 1/5   0   0 1/5   0 1/5 1/5 3/5   0 2/5 2/5 2/5   0 3/5 3/5 1/5   0 2/5 2/5 1/5
0 1/5 1/5 1/5   0 1/5 1/5 2/5   0 1/5 2/5 1/5 1/5;
          0 0 1 0   0 1/5 4/5   0   0 4/5   0 2/5 3/5   0   0 3/5   0 3/5 2/5   0
0 2/5   0 4/5 1/5   0   0 1/5 1/5   0 1/5 3/5 1/5   0 2/5 2/5 1/5   0 3/5 1/5 2/5
0 2/5 1/5 3/5   0 1/5 1/5 2/5   0 1/5 2/5 1/5 1/5 2/5 1/5;
          0 0 0 1   0   0   0 1/5 1/5 1/5   0   0   0 2/5 2/5 2/5   0   0   0 3/5 3/5 3/5
0   0   0 4/5 4/5 4/5   0 1/5 1/5 1/5   0 1/5 1/5 1/5   0 1/5 1/5 1/5   0 2/5 2/5 2/5
0 3/5 3/5 3/5   0 2/5 2/5 2/5 1/5 1/5 1/5 2/5];
 sfun  = 'sf_tet_P5';

 switch i_eval

   case 1

     switch i_dof

       case 1
         vBase = (xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3)*(5*xi(1) - 4))/24;
       case 2
         vBase = (xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3)*(5*xi(2) - 4))/24;
       case 3
         vBase = (xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3)*(5*xi(3) - 4))/24;
       case 4
         vBase = (xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2)*(5*xi(4) - 3)*(5*xi(4) - 4))/24;
       case 5
         vBase = (25*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
       case 6
         vBase = (25*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
       case 7
         vBase = (25*xi(1)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
       case 8
         vBase = (25*xi(1)*xi(4)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
       case 9
         vBase = (25*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
       case 10
         vBase = (25*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
       case 11
```

```
          vBase = (25*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(2) - 1))/12;
        case 12
          vBase = (25*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(3) - 1))/12;
        case 13
          vBase = (25*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1)*(5*xi(3) - 2))/12;
        case 14
          vBase = (25*xi(1)*xi(4)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(4) - 1))/12;
        case 15
          vBase = (25*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(4) - 1))/12;
        case 16
          vBase = (25*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(4) - 1))/12;
        case 17
          vBase = (25*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(2) - 1)*(5*xi(2) - 2))/12;
        case 18
          vBase = (25*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1)*(5*xi(3) - 2))/12;
        case 19
          vBase = (25*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(3) - 1))/12;
        case 20
          vBase = (25*xi(1)*xi(4)*(5*xi(1) - 1)*(5*xi(4) - 1)*(5*xi(4) - 2))/12;
        case 21
          vBase = (25*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(4) - 1)*(5*xi(4) - 2))/12;
        case 22
          vBase = (25*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(4) - 1)*(5*xi(4) - 2))/12;
        case 23
          vBase = (25*xi(1)*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
        case 24
          vBase = (25*xi(2)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
        case 25
          vBase = (25*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
        case 26
          vBase = (25*xi(1)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2)*(5*xi(4) - 3))/24;
        case 27
          vBase = (25*xi(2)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2)*(5*xi(4) - 3))/24;
        case 28
          vBase = (25*xi(3)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2)*(5*xi(4) - 3))/24;
        case 29
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
        case 30
          vBase = (125*xi(1)*xi(2)*xi(4)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
        case 31
          vBase = (125*xi(2)*xi(3)*xi(4)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
        case 32
          vBase = (125*xi(1)*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
        case 33
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(1) - 1)*(5*xi(2) - 1))/4;
        case 34
          vBase = (125*xi(1)*xi(2)*xi(4)*(5*xi(1) - 1)*(5*xi(2) - 1))/4;
        case 35
          vBase = (125*xi(2)*xi(3)*xi(4)*(5*xi(2) - 1)*(5*xi(3) - 1))/4;
        case 36
          vBase = (125*xi(1)*xi(3)*xi(4)*(5*xi(1) - 1)*(5*xi(3) - 1))/4;
        case 37
```

```
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
        case 38
          vBase = (125*xi(1)*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
        case 39
          vBase = (125*xi(2)*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
        case 40
          vBase = (125*xi(1)*xi(3)*xi(4)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
        case 41
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1))/4;
        case 42
          vBase = (125*xi(1)*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(4) - 1))/4;
        case 43
          vBase = (125*xi(2)*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(4) - 1))/4;
        case 44
          vBase = (125*xi(1)*xi(3)*xi(4)*(5*xi(1) - 1)*(5*xi(4) - 1))/4;
        case 45
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
        case 46
          vBase = (125*xi(1)*xi(2)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
        case 47
          vBase = (125*xi(2)*xi(3)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
        case 48
          vBase = (125*xi(1)*xi(3)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
        case 49
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1))/4;
        case 50
          vBase = (125*xi(1)*xi(2)*xi(4)*(5*xi(1) - 1)*(5*xi(4) - 1))/4;
        case 51
          vBase = (125*xi(2)*xi(3)*xi(4)*(5*xi(2) - 1)*(5*xi(4) - 1))/4;
        case 52
          vBase = (125*xi(1)*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(4) - 1))/4;
        case 53
          vBase = (625*xi(1)*xi(2)*xi(3)*xi(4)*(5*xi(1) - 1))/2;
        case 54
          vBase = (625*xi(1)*xi(2)*xi(3)*xi(4)*(5*xi(2) - 1))/2;
        case 55
          vBase = (625*xi(1)*xi(2)*xi(3)*xi(4)*(5*xi(3) - 1))/2;
        case 56
          vBase = (625*xi(1)*xi(2)*xi(3)*xi(4)*(5*xi(4) - 1))/2;
      end

    case {2,3,4}

      switch i_dof

        case 1
          dNdxi1 = (3125*xi(1)^4)/24 - (625*xi(1)^3)/3 + (875*xi(1)^2)/8 - (125*xi(1))/6 + 1;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 2
          dNdxi1 = 0;
```

```
          dNdxi2 = (3125*xi(2)^4)/24 - (625*xi(2)^3)/3 + (875*xi(2)^2)/8 - (125*xi(2))/6 + 1;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 3
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = (3125*xi(3)^4)/24 - (625*xi(3)^3)/3 + (875*xi(3)^2)/8 - (125*xi(3))/6 + 1;
          dNdxi4 = 0;
        case 4
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = (3125*xi(4)^4)/24 - (625*xi(4)^3)/3 + (875*xi(4)^2)/8 - (125*xi(4))/6 + 1;
        case 5
          dNdxi1 = (25*xi(2)*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
          dNdxi2 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 6
          dNdxi1 = 0;
          dNdxi2 = (25*xi(3)*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
          dNdxi3 = (25*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
          dNdxi4 = 0;
        case 7
          dNdxi1 = (25*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
          dNdxi2 = 0;
          dNdxi3 = (25*xi(1)*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
          dNdxi4 = 0;
        case 8
          dNdxi1 = (25*xi(4)*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
          dNdxi2 = 0;
          dNdxi3 = 0;
          dNdxi4 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
        case 9
          dNdxi1 = 0;
          dNdxi2 = (25*xi(4)*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
          dNdxi3 = 0;
          dNdxi4 = (25*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
        case 10
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = (25*xi(4)*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
          dNdxi4 = (25*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
        case 11
          dNdxi1 = (25*xi(2)*(5*xi(2) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
          dNdxi2 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(10*xi(2) - 1))/12;
          dNdxi3 = 0;
          dNdxi4 = 0;
        case 12
          dNdxi1 = 0;
          dNdxi2 = (25*xi(3)*(5*xi(3) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
          dNdxi3 = (25*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(10*xi(3) - 1))/12;
```

```
    dNdxi4 = 0;
  case 13
    dNdxi1 = (25*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1)*(5*xi(3) - 2))/12;
    dNdxi2 = 0;
    dNdxi3 = (25*xi(1)*(5*xi(1) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
    dNdxi4 = 0;
  case 14
    dNdxi1 = (25*xi(4)*(5*xi(4) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
    dNdxi2 = 0;
    dNdxi3 = 0;
    dNdxi4 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(10*xi(4) - 1))/12;
  case 15
    dNdxi1 = 0;
    dNdxi2 = (25*xi(4)*(5*xi(4) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
    dNdxi3 = 0;
    dNdxi4 = (25*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(10*xi(4) - 1))/12;
  case 16
    dNdxi1 = 0;
    dNdxi2 = 0;
    dNdxi3 = (25*xi(4)*(5*xi(4) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
    dNdxi4 = (25*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(10*xi(4) - 1))/12;
  case 17
    dNdxi1 = (25*xi(2)*(10*xi(1) - 1)*(5*xi(2) - 1)*(5*xi(2) - 2))/12;
    dNdxi2 = (25*xi(1)*(5*xi(1) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
    dNdxi3 = 0;
    dNdxi4 = 0;
  case 18
    dNdxi1 = 0;
    dNdxi2 = (25*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1)*(5*xi(3) - 2))/12;
    dNdxi3 = (25*xi(2)*(5*xi(2) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
    dNdxi4 = 0;
  case 19
    dNdxi1 = (25*xi(3)*(5*xi(3) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
    dNdxi2 = 0;
    dNdxi3 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(10*xi(3) - 1))/12;
    dNdxi4 = 0;
  case 20
    dNdxi1 = (25*xi(4)*(10*xi(1) - 1)*(5*xi(4) - 1)*(5*xi(4) - 2))/12;
    dNdxi2 = 0;
    dNdxi3 = 0;
    dNdxi4 = (25*xi(1)*(5*xi(1) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
  case 21
    dNdxi1 = 0;
    dNdxi2 = (25*xi(4)*(10*xi(2) - 1)*(5*xi(4) - 1)*(5*xi(4) - 2))/12;
    dNdxi3 = 0;
    dNdxi4 = (25*xi(2)*(5*xi(2) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
  case 22
    dNdxi1 = 0;
    dNdxi2 = 0;
    dNdxi3 = (25*xi(4)*(10*xi(3) - 1)*(5*xi(4) - 1)*(5*xi(4) - 2))/12;
    dNdxi4 = (25*xi(3)*(5*xi(3) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
  case 23
```

```
        dNdxi1 = (25*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
        dNdxi2 = (25*xi(1)*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
        dNdxi3 = 0;
        dNdxi4 = 0;
      case 24
        dNdxi1 = 0;
        dNdxi2 = (25*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
        dNdxi3 = (25*xi(2)*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
        dNdxi4 = 0;
      case 25
        dNdxi1 = (25*xi(3)*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
        dNdxi2 = 0;
        dNdxi3 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
        dNdxi4 = 0;
      case 26
        dNdxi1 = (25*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2)*(5*xi(4) - 3))/24;
        dNdxi2 = 0;
        dNdxi3 = 0;
        dNdxi4 = (25*xi(1)*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
      case 27
        dNdxi1 = 0;
        dNdxi2 = (25*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2)*(5*xi(4) - 3))/24;
        dNdxi3 = 0;
        dNdxi4 = (25*xi(2)*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
      case 28
        dNdxi1 = 0;
        dNdxi2 = 0;
        dNdxi3 = (25*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2)*(5*xi(4) - 3))/24;
        dNdxi4 = (25*xi(3)*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
      case 29
        dNdxi1 = (125*xi(2)*xi(3)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
        dNdxi2 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
        dNdxi3 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
        dNdxi4 = 0;
      case 30
        dNdxi1 = (125*xi(2)*xi(4)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
        dNdxi2 = (125*xi(1)*xi(4)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
        dNdxi3 = 0;
        dNdxi4 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
      case 31
        dNdxi1 = 0;
        dNdxi2 = (125*xi(3)*xi(4)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
        dNdxi3 = (125*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
        dNdxi4 = (125*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
      case 32
        dNdxi1 = (125*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
        dNdxi2 = 0;
        dNdxi3 = (125*xi(1)*xi(4)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
        dNdxi4 = (125*xi(1)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
      case 33
        dNdxi1 = (125*xi(2)*xi(3)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
        dNdxi2 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
```

```
      dNdxi3 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(2) - 1))/4;
      dNdxi4 = 0;
    case 34
      dNdxi1 = (125*xi(2)*xi(4)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
      dNdxi2 = (125*xi(1)*xi(4)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
      dNdxi3 = 0;
      dNdxi4 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(2) - 1))/4;
    case 35
      dNdxi1 = 0;
      dNdxi2 = (125*xi(3)*xi(4)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
      dNdxi3 = (125*xi(2)*xi(4)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
      dNdxi4 = (125*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1))/4;
    case 36
      dNdxi1 = (125*xi(3)*xi(4)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
      dNdxi2 = 0;
      dNdxi3 = (125*xi(1)*xi(4)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
      dNdxi4 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1))/4;
    case 37
      dNdxi1 = (125*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
      dNdxi2 = (125*xi(1)*xi(3)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
      dNdxi3 = (125*xi(1)*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
      dNdxi4 = 0;
    case 38
      dNdxi1 = (125*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
      dNdxi2 = (125*xi(1)*xi(4)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
      dNdxi3 = 0;
      dNdxi4 = (125*xi(1)*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
    case 39
      dNdxi1 = 0;
      dNdxi2 = (125*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
      dNdxi3 = (125*xi(2)*xi(4)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
      dNdxi4 = (125*xi(2)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
    case 40
      dNdxi1 = (125*xi(3)*xi(4)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
      dNdxi2 = 0;
      dNdxi3 = (125*xi(1)*xi(4)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
      dNdxi4 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    case 41
      dNdxi1 = (125*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1))/4;
      dNdxi2 = (125*xi(1)*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
      dNdxi3 = (125*xi(1)*xi(2)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
      dNdxi4 = 0;
    case 42
      dNdxi1 = (125*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(4) - 1))/4;
      dNdxi2 = (125*xi(1)*xi(4)*(10*xi(2) - 1)*(5*xi(4) - 1))/4;
      dNdxi3 = 0;
      dNdxi4 = (125*xi(1)*xi(2)*(5*xi(2) - 1)*(10*xi(4) - 1))/4;
    case 43
      dNdxi1 = 0;
      dNdxi2 = (125*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(4) - 1))/4;
      dNdxi3 = (125*xi(2)*xi(4)*(10*xi(3) - 1)*(5*xi(4) - 1))/4;
      dNdxi4 = (125*xi(2)*xi(3)*(5*xi(3) - 1)*(10*xi(4) - 1))/4;
```

```
        case 44
          dNdxi1 = (125*xi(3)*xi(4)*(10*xi(1) - 1)*(5*xi(4) - 1))/4;
          dNdxi2 = 0;
          dNdxi3 = (125*xi(1)*xi(4)*(5*xi(1) - 1)*(5*xi(4) - 1))/4;
          dNdxi4 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(10*xi(4) - 1))/4;
        case 45
          dNdxi1 = (125*xi(2)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          dNdxi2 = (125*xi(1)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          dNdxi3 = (125*xi(1)*xi(2)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
          dNdxi4 = 0;
        case 46
          dNdxi1 = (125*xi(2)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
          dNdxi2 = (125*xi(1)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
          dNdxi3 = 0;
          dNdxi4 = (125*xi(1)*xi(2)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
        case 47
          dNdxi1 = 0;
          dNdxi2 = (125*xi(3)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
          dNdxi3 = (125*xi(2)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
          dNdxi4 = (125*xi(2)*xi(3)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
        case 48
          dNdxi1 = (125*xi(3)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
          dNdxi2 = 0;
          dNdxi3 = (125*xi(1)*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
          dNdxi4 = (125*xi(1)*xi(3)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
        case 49
          dNdxi1 = (125*xi(2)*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
          dNdxi2 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1))/4;
          dNdxi3 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
          dNdxi4 = 0;
        case 50
          dNdxi1 = (125*xi(2)*xi(4)*(10*xi(1) - 1)*(5*xi(4) - 1))/4;
          dNdxi2 = (125*xi(1)*xi(4)*(5*xi(1) - 1)*(5*xi(4) - 1))/4;
          dNdxi3 = 0;
          dNdxi4 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(10*xi(4) - 1))/4;
        case 51
          dNdxi1 = 0;
          dNdxi2 = (125*xi(3)*xi(4)*(10*xi(2) - 1)*(5*xi(4) - 1))/4;
          dNdxi3 = (125*xi(2)*xi(4)*(5*xi(2) - 1)*(5*xi(4) - 1))/4;
          dNdxi4 = (125*xi(2)*xi(3)*(5*xi(2) - 1)*(10*xi(4) - 1))/4;
        case 52
          dNdxi1 = (125*xi(3)*xi(4)*(5*xi(3) - 1)*(5*xi(4) - 1))/4;
          dNdxi2 = 0;
          dNdxi3 = (125*xi(1)*xi(4)*(10*xi(3) - 1)*(5*xi(4) - 1))/4;
          dNdxi4 = (125*xi(1)*xi(3)*(5*xi(3) - 1)*(10*xi(4) - 1))/4;
        case 53
          dNdxi1 = (625*xi(2)*xi(3)*xi(4)*(10*xi(1) - 1))/2;
          dNdxi2 = (625*xi(1)*xi(3)*xi(4)*(5*xi(1) - 1))/2;
          dNdxi3 = (625*xi(1)*xi(2)*xi(4)*(5*xi(1) - 1))/2;
          dNdxi4 = (625*xi(1)*xi(2)*xi(3)*(5*xi(1) - 1))/2;
        case 54
          dNdxi1 = (625*xi(2)*xi(3)*xi(4)*(5*xi(2) - 1))/2;
```

```
          dNdxi2 = (625*xi(1)*xi(3)*xi(4)*(10*xi(2) - 1))/2;
          dNdxi3 = (625*xi(1)*xi(2)*xi(4)*(5*xi(2) - 1))/2;
          dNdxi4 = (625*xi(1)*xi(2)*xi(3)*(5*xi(2) - 1))/2;
        case 55
          dNdxi1 = (625*xi(2)*xi(3)*xi(4)*(5*xi(3) - 1))/2;
          dNdxi2 = (625*xi(1)*xi(3)*xi(4)*(5*xi(3) - 1))/2;
          dNdxi3 = (625*xi(1)*xi(2)*xi(4)*(10*xi(3) - 1))/2;
          dNdxi4 = (625*xi(1)*xi(2)*xi(3)*(5*xi(3) - 1))/2;
        case 56
          dNdxi1 = (625*xi(2)*xi(3)*xi(4)*(5*xi(4) - 1))/2;
          dNdxi2 = (625*xi(1)*xi(3)*xi(4)*(5*xi(4) - 1))/2;
          dNdxi3 = (625*xi(1)*xi(2)*xi(4)*(5*xi(4) - 1))/2;
          dNdxi4 = (625*xi(1)*xi(2)*xi(3)*(10*xi(4) - 1))/2;
      end

      if( i_eval==2 )

        vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3 + aInvJac(:,4)*dNdx

      elseif( i_eval==3 )

        vBase = aInvJac(:,5)*dNdxi1 + aInvJac(:,6)*dNdxi2 + aInvJac(:,7)*dNdxi3 + aInvJac(:,8)*dNdx

      else

        vBase = aInvJac(:,9)*dNdxi1 + aInvJac(:,10)*dNdxi2 + aInvJac(:,11)*dNdxi3 + aInvJac(:,12)*d

      end

    case {22,23,24,32,33,34,42,43,44}   % Evaluation of second derivatives.

      switch i_dof

        case 1
          d2Ndxi1dxi1 = (125*(5*xi(1) - 2)*(10*xi(1)^2 - 8*xi(1) + 1))/12;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 2
```

```
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = (125*(5*xi(2) - 2)*(10*xi(2)^2 - 8*xi(2) + 1))/12;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 3
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = (125*(5*xi(3) - 2)*(10*xi(3)^2 - 8*xi(3) + 1))/12;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 4
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = (125*(5*xi(4) - 2)*(10*xi(4)^2 - 8*xi(4) + 1))/12;
```

```
        case 5
          d2Ndxi1dxi1 = (125*xi(2)*(150*xi(1)^2 - 90*xi(1) + 11))/12;
          d2Ndxi2dxi1 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 6
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = (125*xi(3)*(150*xi(2)^2 - 90*xi(2) + 11))/12;
          d2Ndxi3dxi2 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 7
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = (125*xi(1)*(150*xi(3)^2 - 90*xi(3) + 11))/12;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
```

```
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

    case 8
        d2Ndxi1dxi1 = (125*xi(4)*(150*xi(1)^2 - 90*xi(1) + 11))/12;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

    case 9
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = (125*xi(4)*(150*xi(2)^2 - 90*xi(2) + 11))/12;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

    case 10
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = (125*xi(4)*(150*xi(3)^2 - 90*xi(3) + 11))/12;
        d2Ndxi4dxi3 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
```

```
      d2Ndxi1dxi4 = 0;
      d2Ndxi2dxi4 = 0;
      d2Ndxi3dxi4 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
      d2Ndxi4dxi4 = 0;

    case 11
      d2Ndxi1dxi1 = (125*xi(2)*(5*xi(1) - 1)*(5*xi(2) - 1))/2;
      d2Ndxi2dxi1 = (25*(10*xi(2) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
      d2Ndxi3dxi1 = 0;
      d2Ndxi4dxi1 = 0;
      d2Ndxi1dxi2 = (25*(10*xi(2) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
      d2Ndxi2dxi2 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
      d2Ndxi3dxi2 = 0;
      d2Ndxi4dxi2 = 0;
      d2Ndxi1dxi3 = 0;
      d2Ndxi2dxi3 = 0;
      d2Ndxi3dxi3 = 0;
      d2Ndxi4dxi3 = 0;
      d2Ndxi1dxi4 = 0;
      d2Ndxi2dxi4 = 0;
      d2Ndxi3dxi4 = 0;
      d2Ndxi4dxi4 = 0;

    case 12
      d2Ndxi1dxi1 = 0;
      d2Ndxi2dxi1 = 0;
      d2Ndxi3dxi1 = 0;
      d2Ndxi4dxi1 = 0;
      d2Ndxi1dxi2 = 0;
      d2Ndxi2dxi2 = (125*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1))/2;
      d2Ndxi3dxi2 = (25*(10*xi(3) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
      d2Ndxi4dxi2 = 0;
      d2Ndxi1dxi3 = 0;
      d2Ndxi2dxi3 = (25*(10*xi(3) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
      d2Ndxi3dxi3 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
      d2Ndxi4dxi3 = 0;
      d2Ndxi1dxi4 = 0;
      d2Ndxi2dxi4 = 0;
      d2Ndxi3dxi4 = 0;
      d2Ndxi4dxi4 = 0;

    case 13
      d2Ndxi1dxi1 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
      d2Ndxi2dxi1 = 0;
      d2Ndxi3dxi1 = (25*(10*xi(1) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
      d2Ndxi4dxi1 = 0;
      d2Ndxi1dxi2 = 0;
      d2Ndxi2dxi2 = 0;
      d2Ndxi3dxi2 = 0;
      d2Ndxi4dxi2 = 0;
      d2Ndxi1dxi3 = (25*(10*xi(1) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
      d2Ndxi2dxi3 = 0;
```

```
    d2Ndxi3dxi3 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(3) - 1))/2;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 14
    d2Ndxi1dxi1 = (125*xi(4)*(5*xi(1) - 1)*(5*xi(4) - 1))/2;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = (25*(10*xi(4) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = (25*(10*xi(4) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;

  case 15
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = (125*xi(4)*(5*xi(2) - 1)*(5*xi(4) - 1))/2;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = (25*(10*xi(4) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = (25*(10*xi(4) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;

  case 16
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
```

```
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = (125*xi(4)*(5*xi(3) - 1)*(5*xi(4) - 1))/2;
          d2Ndxi4dxi3 = (25*(10*xi(4) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = (25*(10*xi(4) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
          d2Ndxi4dxi4 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;

        case 17
          d2Ndxi1dxi1 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
          d2Ndxi2dxi1 = (25*(10*xi(1) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = (25*(10*xi(1) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
          d2Ndxi2dxi2 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(2) - 1))/2;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 18
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          d2Ndxi3dxi2 = (25*(10*xi(2) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = (25*(10*xi(2) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
          d2Ndxi3dxi3 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(3) - 1))/2;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 19
          d2Ndxi1dxi1 = (125*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1))/2;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = (25*(10*xi(3) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
```

```
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = (25*(10*xi(3) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

      case 20
        d2Ndxi1dxi1 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = (25*(10*xi(1) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = (25*(10*xi(1) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(4) - 1))/2;

      case 21
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = (25*(10*xi(2) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = (25*(10*xi(2) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(4) - 1))/2;

      case 22
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
```

```
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
    d2Ndxi4dxi3 = (25*(10*xi(3) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = (25*(10*xi(3) - 1)*(75*xi(4)^2 - 30*xi(4) + 2))/12;
    d2Ndxi4dxi4 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(4) - 1))/2;

  case 23
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
    d2Ndxi2dxi2 = (125*xi(1)*(150*xi(2)^2 - 90*xi(2) + 11))/12;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 24
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
    d2Ndxi3dxi3 = (125*xi(2)*(150*xi(3)^2 - 90*xi(3) + 11))/12;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 25
    d2Ndxi1dxi1 = (125*xi(3)*(150*xi(1)^2 - 90*xi(1) + 11))/12;
    d2Ndxi2dxi1 = 0;
```

```
        d2Ndxi3dxi1 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = 0;

      case 26
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = (25*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = (25*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
        d2Ndxi2dxi4 = 0;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = (125*xi(1)*(150*xi(4)^2 - 90*xi(4) + 11))/12;

      case 27
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi4dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi4dxi2 = (25*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;
        d2Ndxi4dxi3 = 0;
        d2Ndxi1dxi4 = 0;
        d2Ndxi2dxi4 = (25*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
        d2Ndxi3dxi4 = 0;
        d2Ndxi4dxi4 = (125*xi(2)*(150*xi(4)^2 - 90*xi(4) + 11))/12;

      case 28
```

```
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = (25*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = (25*(10*xi(4) - 3)*(25*xi(4)^2 - 15*xi(4) + 1))/12;
    d2Ndxi4dxi4 = (125*xi(3)*(150*xi(4)^2 - 90*xi(4) + 11))/12;

  case 29
    d2Ndxi1dxi1 = 625*xi(2)*xi(3)*(5*xi(1) - 1);
    d2Ndxi2dxi1 = (125*xi(3)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi3dxi1 = (125*xi(2)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = (125*xi(3)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = (125*xi(2)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi2dxi3 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 30
    d2Ndxi1dxi1 = 625*xi(2)*xi(4)*(5*xi(1) - 1);
    d2Ndxi2dxi1 = (125*xi(4)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = (125*xi(2)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi1dxi2 = (125*xi(4)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = (125*xi(2)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi2dxi4 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;
```

```
        case 31
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 625*xi(3)*xi(4)*(5*xi(2) - 1);
          d2Ndxi3dxi2 = (125*xi(4)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
          d2Ndxi4dxi2 = (125*xi(3)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = (125*xi(4)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = (125*xi(3)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
          d2Ndxi3dxi4 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
          d2Ndxi4dxi4 = 0;

        case 32
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = (125*xi(4)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
          d2Ndxi4dxi1 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = (125*xi(4)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 625*xi(1)*xi(4)*(5*xi(3) - 1);
          d2Ndxi4dxi3 = (125*xi(1)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
          d2Ndxi1dxi4 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = (125*xi(1)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
          d2Ndxi4dxi4 = 0;

        case 33
          d2Ndxi1dxi1 = (625*xi(2)*xi(3)*(5*xi(2) - 1))/2;
          d2Ndxi2dxi1 = (125*xi(3)*(10*xi(1) - 1)*(10*xi(2) - 1))/4;
          d2Ndxi3dxi1 = (125*xi(2)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = (125*xi(3)*(10*xi(1) - 1)*(10*xi(2) - 1))/4;
          d2Ndxi2dxi2 = (625*xi(1)*xi(3)*(5*xi(1) - 1))/2;
          d2Ndxi3dxi2 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = (125*xi(2)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
          d2Ndxi2dxi3 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
```

```
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 34
    d2Ndxi1dxi1 = (625*xi(2)*xi(4)*(5*xi(2) - 1))/2;
    d2Ndxi2dxi1 = (125*xi(4)*(10*xi(1) - 1)*(10*xi(2) - 1))/4;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = (125*xi(2)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
    d2Ndxi1dxi2 = (125*xi(4)*(10*xi(1) - 1)*(10*xi(2) - 1))/4;
    d2Ndxi2dxi2 = (625*xi(1)*xi(4)*(5*xi(1) - 1))/2;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = (125*xi(2)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
    d2Ndxi2dxi4 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 35
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = (625*xi(3)*xi(4)*(5*xi(3) - 1))/2;
    d2Ndxi3dxi2 = (125*xi(4)*(10*xi(2) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi4dxi2 = (125*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = (125*xi(4)*(10*xi(2) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi3dxi3 = (625*xi(2)*xi(4)*(5*xi(2) - 1))/2;
    d2Ndxi4dxi3 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = (125*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
    d2Ndxi3dxi4 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi4dxi4 = 0;

  case 36
    d2Ndxi1dxi1 = (625*xi(3)*xi(4)*(5*xi(3) - 1))/2;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = (125*xi(4)*(10*xi(1) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi4dxi1 = (125*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = (125*xi(4)*(10*xi(1) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = (625*xi(1)*xi(4)*(5*xi(1) - 1))/2;
    d2Ndxi4dxi3 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
```

```
    d2Ndxi1dxi4 = (125*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi4dxi4 = 0;

  case 37
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (125*xi(3)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi3dxi1 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = (125*xi(3)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi2dxi2 = 625*xi(1)*xi(3)*(5*xi(2) - 1);
    d2Ndxi3dxi2 = (125*xi(1)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
    d2Ndxi2dxi3 = (125*xi(1)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 38
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (125*xi(4)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
    d2Ndxi1dxi2 = (125*xi(4)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi2dxi2 = 625*xi(1)*xi(4)*(5*xi(2) - 1);
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = (125*xi(1)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
    d2Ndxi2dxi4 = (125*xi(1)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 39
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = (125*xi(4)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
    d2Ndxi4dxi2 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = (125*xi(4)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
```

```
    d2Ndxi3dxi3 = 625*xi(2)*xi(4)*(5*xi(3) - 1);
    d2Ndxi4dxi3 = (125*xi(2)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
    d2Ndxi3dxi4 = (125*xi(2)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
    d2Ndxi4dxi4 = 0;

  case 40
    d2Ndxi1dxi1 = 625*xi(3)*xi(4)*(5*xi(1) - 1);
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = (125*xi(4)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi4dxi1 = (125*xi(3)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = (125*xi(4)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi1dxi4 = (125*xi(3)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi4dxi4 = 0;

  case 41
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (125*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
    d2Ndxi3dxi1 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = (125*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
    d2Ndxi2dxi2 = (625*xi(1)*xi(3)*(5*xi(3) - 1))/2;
    d2Ndxi3dxi2 = (125*xi(1)*(10*xi(2) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi2dxi3 = (125*xi(1)*(10*xi(2) - 1)*(10*xi(3) - 1))/4;
    d2Ndxi3dxi3 = (625*xi(1)*xi(2)*(5*xi(2) - 1))/2;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 42
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (125*xi(4)*(10*xi(2) - 1)*(5*xi(4) - 1))/4;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(4) - 1))/4;
    d2Ndxi1dxi2 = (125*xi(4)*(10*xi(2) - 1)*(5*xi(4) - 1))/4;
    d2Ndxi2dxi2 = (625*xi(1)*xi(4)*(5*xi(4) - 1))/2;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = (125*xi(1)*(10*xi(2) - 1)*(10*xi(4) - 1))/4;
```

```
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi2dxi4 = (125*xi(1)*(10*xi(2) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = (625*xi(1)*xi(2)*(5*xi(2) - 1))/2;

        case 43
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = (125*xi(4)*(10*xi(3) - 1)*(5*xi(4) - 1))/4;
          d2Ndxi4dxi2 = (125*xi(3)*(5*xi(3) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = (125*xi(4)*(10*xi(3) - 1)*(5*xi(4) - 1))/4;
          d2Ndxi3dxi3 = (625*xi(2)*xi(4)*(5*xi(4) - 1))/2;
          d2Ndxi4dxi3 = (125*xi(2)*(10*xi(3) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = (125*xi(3)*(5*xi(3) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi3dxi4 = (125*xi(2)*(10*xi(3) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi4dxi4 = (625*xi(2)*xi(3)*(5*xi(3) - 1))/2;

        case 44
          d2Ndxi1dxi1 = (625*xi(3)*xi(4)*(5*xi(4) - 1))/2;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = (125*xi(4)*(10*xi(1) - 1)*(5*xi(4) - 1))/4;
          d2Ndxi4dxi1 = (125*xi(3)*(10*xi(1) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = (125*xi(4)*(10*xi(1) - 1)*(5*xi(4) - 1))/4;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi1dxi4 = (125*xi(3)*(10*xi(1) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi4dxi4 = (625*xi(1)*xi(3)*(5*xi(1) - 1))/2;

        case 45
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          d2Ndxi3dxi1 = (125*xi(2)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          d2Ndxi2dxi2 = 0;
```

```
    d2Ndxi3dxi2 = (125*xi(1)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
    d2Ndxi4dxi2 = 0;
    d2Ndxi1dxi3 = (125*xi(2)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
    d2Ndxi2dxi3 = (125*xi(1)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
    d2Ndxi3dxi3 = 625*xi(1)*xi(2)*(5*xi(3) - 1);
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = 0;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 0;

  case 46
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = (125*xi(2)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
    d2Ndxi1dxi2 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi4dxi2 = (125*xi(1)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = 0;
    d2Ndxi1dxi4 = (125*xi(2)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
    d2Ndxi2dxi4 = (125*xi(1)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
    d2Ndxi3dxi4 = 0;
    d2Ndxi4dxi4 = 625*xi(1)*xi(2)*(5*xi(4) - 1);

  case 47
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi4dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
    d2Ndxi4dxi2 = (125*xi(3)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = (125*xi(2)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
    d2Ndxi1dxi4 = 0;
    d2Ndxi2dxi4 = (125*xi(3)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
    d2Ndxi3dxi4 = (125*xi(2)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
    d2Ndxi4dxi4 = 625*xi(2)*xi(3)*(5*xi(4) - 1);

  case 48
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
    d2Ndxi4dxi1 = (125*xi(3)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
```

```
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = (125*xi(4)*(5*xi(4) - 1)*(5*xi(4) - 2))/6;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = (125*xi(1)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
          d2Ndxi1dxi4 = (125*xi(3)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = (125*xi(1)*(75*xi(4)^2 - 30*xi(4) + 2))/6;
          d2Ndxi4dxi4 = 625*xi(1)*xi(3)*(5*xi(4) - 1);

        case 49
          d2Ndxi1dxi1 = (625*xi(2)*xi(3)*(5*xi(3) - 1))/2;
          d2Ndxi2dxi1 = (125*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
          d2Ndxi3dxi1 = (125*xi(2)*(10*xi(1) - 1)*(10*xi(3) - 1))/4;
          d2Ndxi4dxi1 = 0;
          d2Ndxi1dxi2 = (125*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
          d2Ndxi4dxi2 = 0;
          d2Ndxi1dxi3 = (125*xi(2)*(10*xi(1) - 1)*(10*xi(3) - 1))/4;
          d2Ndxi2dxi3 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
          d2Ndxi3dxi3 = (625*xi(1)*xi(2)*(5*xi(1) - 1))/2;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = 0;
          d2Ndxi2dxi4 = 0;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = 0;

        case 50
          d2Ndxi1dxi1 = (625*xi(2)*xi(4)*(5*xi(4) - 1))/2;
          d2Ndxi2dxi1 = (125*xi(4)*(10*xi(1) - 1)*(5*xi(4) - 1))/4;
          d2Ndxi3dxi1 = 0;
          d2Ndxi4dxi1 = (125*xi(2)*(10*xi(1) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi1dxi2 = (125*xi(4)*(10*xi(1) - 1)*(5*xi(4) - 1))/4;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi4dxi2 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;
          d2Ndxi4dxi3 = 0;
          d2Ndxi1dxi4 = (125*xi(2)*(10*xi(1) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi2dxi4 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(4) - 1))/4;
          d2Ndxi3dxi4 = 0;
          d2Ndxi4dxi4 = (625*xi(1)*xi(2)*(5*xi(1) - 1))/2;

        case 51
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
```

```
      d2Ndxi3dxi1 = 0;
      d2Ndxi4dxi1 = 0;
      d2Ndxi1dxi2 = 0;
      d2Ndxi2dxi2 = (625*xi(3)*xi(4)*(5*xi(4) - 1))/2;
      d2Ndxi3dxi2 = (125*xi(4)*(10*xi(2) - 1)*(5*xi(4) - 1))/4;
      d2Ndxi4dxi2 = (125*xi(3)*(10*xi(2) - 1)*(10*xi(4) - 1))/4;
      d2Ndxi1dxi3 = 0;
      d2Ndxi2dxi3 = (125*xi(4)*(10*xi(2) - 1)*(5*xi(4) - 1))/4;
      d2Ndxi3dxi3 = 0;
      d2Ndxi4dxi3 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(4) - 1))/4;
      d2Ndxi1dxi4 = 0;
      d2Ndxi2dxi4 = (125*xi(3)*(10*xi(2) - 1)*(10*xi(4) - 1))/4;
      d2Ndxi3dxi4 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(4) - 1))/4;
      d2Ndxi4dxi4 = (625*xi(2)*xi(3)*(5*xi(2) - 1))/2;

    case 52
      d2Ndxi1dxi1 = 0;
      d2Ndxi2dxi1 = 0;
      d2Ndxi3dxi1 = (125*xi(4)*(10*xi(3) - 1)*(5*xi(4) - 1))/4;
      d2Ndxi4dxi1 = (125*xi(3)*(5*xi(3) - 1)*(10*xi(4) - 1))/4;
      d2Ndxi1dxi2 = 0;
      d2Ndxi2dxi2 = 0;
      d2Ndxi3dxi2 = 0;
      d2Ndxi4dxi2 = 0;
      d2Ndxi1dxi3 = (125*xi(4)*(10*xi(3) - 1)*(5*xi(4) - 1))/4;
      d2Ndxi2dxi3 = 0;
      d2Ndxi3dxi3 = (625*xi(1)*xi(4)*(5*xi(4) - 1))/2;
      d2Ndxi4dxi3 = (125*xi(1)*(10*xi(3) - 1)*(10*xi(4) - 1))/4;
      d2Ndxi1dxi4 = (125*xi(3)*(5*xi(3) - 1)*(10*xi(4) - 1))/4;
      d2Ndxi2dxi4 = 0;
      d2Ndxi3dxi4 = (125*xi(1)*(10*xi(3) - 1)*(10*xi(4) - 1))/4;
      d2Ndxi4dxi4 = (625*xi(1)*xi(3)*(5*xi(3) - 1))/2;

    case 53
      d2Ndxi1dxi1 = 3125*xi(2)*xi(3)*xi(4);
      d2Ndxi2dxi1 = (625*xi(3)*xi(4)*(10*xi(1) - 1))/2;
      d2Ndxi3dxi1 = (625*xi(2)*xi(4)*(10*xi(1) - 1))/2;
      d2Ndxi4dxi1 = (625*xi(2)*xi(3)*(10*xi(1) - 1))/2;
      d2Ndxi1dxi2 = (625*xi(3)*xi(4)*(10*xi(1) - 1))/2;
      d2Ndxi2dxi2 = 0;
      d2Ndxi3dxi2 = (625*xi(1)*xi(4)*(5*xi(1) - 1))/2;
      d2Ndxi4dxi2 = (625*xi(1)*xi(3)*(5*xi(1) - 1))/2;
      d2Ndxi1dxi3 = (625*xi(2)*xi(4)*(10*xi(1) - 1))/2;
      d2Ndxi2dxi3 = (625*xi(1)*xi(4)*(5*xi(1) - 1))/2;
      d2Ndxi3dxi3 = 0;
      d2Ndxi4dxi3 = (625*xi(1)*xi(2)*(5*xi(1) - 1))/2;
      d2Ndxi1dxi4 = (625*xi(2)*xi(3)*(10*xi(1) - 1))/2;
      d2Ndxi2dxi4 = (625*xi(1)*xi(3)*(5*xi(1) - 1))/2;
      d2Ndxi3dxi4 = (625*xi(1)*xi(2)*(5*xi(1) - 1))/2;
      d2Ndxi4dxi4 = 0;

    case 54
```

```
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (625*xi(3)*xi(4)*(10*xi(2) - 1))/2;
    d2Ndxi3dxi1 = (625*xi(2)*xi(4)*(5*xi(2) - 1))/2;
    d2Ndxi4dxi1 = (625*xi(2)*xi(3)*(5*xi(2) - 1))/2;
    d2Ndxi1dxi2 = (625*xi(3)*xi(4)*(10*xi(2) - 1))/2;
    d2Ndxi2dxi2 = 3125*xi(1)*xi(3)*xi(4);
    d2Ndxi3dxi2 = (625*xi(1)*xi(4)*(10*xi(2) - 1))/2;
    d2Ndxi4dxi2 = (625*xi(1)*xi(3)*(10*xi(2) - 1))/2;
    d2Ndxi1dxi3 = (625*xi(2)*xi(4)*(5*xi(2) - 1))/2;
    d2Ndxi2dxi3 = (625*xi(1)*xi(4)*(10*xi(2) - 1))/2;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = (625*xi(1)*xi(2)*(5*xi(2) - 1))/2;
    d2Ndxi1dxi4 = (625*xi(2)*xi(3)*(5*xi(2) - 1))/2;
    d2Ndxi2dxi4 = (625*xi(1)*xi(3)*(10*xi(2) - 1))/2;
    d2Ndxi3dxi4 = (625*xi(1)*xi(2)*(5*xi(2) - 1))/2;
    d2Ndxi4dxi4 = 0;

  case 55
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (625*xi(3)*xi(4)*(5*xi(3) - 1))/2;
    d2Ndxi3dxi1 = (625*xi(2)*xi(4)*(10*xi(3) - 1))/2;
    d2Ndxi4dxi1 = (625*xi(2)*xi(3)*(5*xi(3) - 1))/2;
    d2Ndxi1dxi2 = (625*xi(3)*xi(4)*(5*xi(3) - 1))/2;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = (625*xi(1)*xi(4)*(10*xi(3) - 1))/2;
    d2Ndxi4dxi2 = (625*xi(1)*xi(3)*(5*xi(3) - 1))/2;
    d2Ndxi1dxi3 = (625*xi(2)*xi(4)*(10*xi(3) - 1))/2;
    d2Ndxi2dxi3 = (625*xi(1)*xi(4)*(10*xi(3) - 1))/2;
    d2Ndxi3dxi3 = 3125*xi(1)*xi(2)*xi(4);
    d2Ndxi4dxi3 = (625*xi(1)*xi(2)*(10*xi(3) - 1))/2;
    d2Ndxi1dxi4 = (625*xi(2)*xi(3)*(5*xi(3) - 1))/2;
    d2Ndxi2dxi4 = (625*xi(1)*xi(3)*(5*xi(3) - 1))/2;
    d2Ndxi3dxi4 = (625*xi(1)*xi(2)*(10*xi(3) - 1))/2;
    d2Ndxi4dxi4 = 0;

  case 56
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (625*xi(3)*xi(4)*(5*xi(4) - 1))/2;
    d2Ndxi3dxi1 = (625*xi(2)*xi(4)*(5*xi(4) - 1))/2;
    d2Ndxi4dxi1 = (625*xi(2)*xi(3)*(10*xi(4) - 1))/2;
    d2Ndxi1dxi2 = (625*xi(3)*xi(4)*(5*xi(4) - 1))/2;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = (625*xi(1)*xi(4)*(5*xi(4) - 1))/2;
    d2Ndxi4dxi2 = (625*xi(1)*xi(3)*(10*xi(4) - 1))/2;
    d2Ndxi1dxi3 = (625*xi(2)*xi(4)*(5*xi(4) - 1))/2;
    d2Ndxi2dxi3 = (625*xi(1)*xi(4)*(5*xi(4) - 1))/2;
    d2Ndxi3dxi3 = 0;
    d2Ndxi4dxi3 = (625*xi(1)*xi(2)*(10*xi(4) - 1))/2;
    d2Ndxi1dxi4 = (625*xi(2)*xi(3)*(10*xi(4) - 1))/2;
    d2Ndxi2dxi4 = (625*xi(1)*xi(3)*(10*xi(4) - 1))/2;
    d2Ndxi3dxi4 = (625*xi(1)*xi(2)*(10*xi(4) - 1))/2;
    d2Ndxi4dxi4 = 3125*xi(1)*xi(2)*xi(3);
```

```
    end

    switch( i_eval )
      case 22
        vBase = aInvJac(:,1).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:,
                aInvJac(:,2).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:,
                aInvJac(:,3).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:,
                aInvJac(:,4).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:,
      case 33
        vBase = aInvJac(:,5).*( aInvJac(:,5)*d2Ndxi1dxi1 + aInvJac(:, 6)*d2Ndxi2dxi1 + aInvJac(:,
                aInvJac(:,6).*( aInvJac(:,5)*d2Ndxi1dxi2 + aInvJac(:, 6)*d2Ndxi2dxi2 + aInvJac(:,
                aInvJac(:,7).*( aInvJac(:,5)*d2Ndxi1dxi3 + aInvJac(:, 6)*d2Ndxi2dxi3 + aInvJac(:,
                aInvJac(:,8).*( aInvJac(:,5)*d2Ndxi1dxi4 + aInvJac(:, 6)*d2Ndxi2dxi4 + aInvJac(:,

      case 44
        vBase = aInvJac(:, 9).*( aInvJac(:,9)*d2Ndxi1dxi1 + aInvJac(:,10)*d2Ndxi2dxi1 + aInvJac(:
                aInvJac(:,10).*( aInvJac(:,9)*d2Ndxi1dxi2 + aInvJac(:,10)*d2Ndxi2dxi2 + aInvJac(:
                aInvJac(:,11).*( aInvJac(:,9)*d2Ndxi1dxi3 + aInvJac(:,10)*d2Ndxi2dxi3 + aInvJac(:
                aInvJac(:,12).*( aInvJac(:,9)*d2Ndxi1dxi4 + aInvJac(:,10)*d2Ndxi2dxi4 + aInvJac(:

      case {23,32}
        vBase = aInvJac(:,5).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:,
                aInvJac(:,6).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:,
                aInvJac(:,7).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:,
                aInvJac(:,8).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:,

      case {24,42}
        vBase = aInvJac(:, 9).*( aInvJac(:,1)*d2Ndxi1dxi1 + aInvJac(:, 2)*d2Ndxi2dxi1 + aInvJac(:
                aInvJac(:,10).*( aInvJac(:,1)*d2Ndxi1dxi2 + aInvJac(:, 2)*d2Ndxi2dxi2 + aInvJac(:
                aInvJac(:,11).*( aInvJac(:,1)*d2Ndxi1dxi3 + aInvJac(:, 2)*d2Ndxi2dxi3 + aInvJac(:
                aInvJac(:,12).*( aInvJac(:,1)*d2Ndxi1dxi4 + aInvJac(:, 2)*d2Ndxi2dxi4 + aInvJac(:

      case {34,43}
        vBase = aInvJac(:, 9).*( aInvJac(:,5)*d2Ndxi1dxi1 + aInvJac(:, 6)*d2Ndxi2dxi1 + aInvJac(:
                aInvJac(:,10).*( aInvJac(:,5)*d2Ndxi1dxi2 + aInvJac(:, 6)*d2Ndxi2dxi2 + aInvJac(:
                aInvJac(:,11).*( aInvJac(:,5)*d2Ndxi1dxi3 + aInvJac(:, 6)*d2Ndxi2dxi3 + aInvJac(:
                aInvJac(:,12).*( aInvJac(:,5)*d2Ndxi1dxi4 + aInvJac(:, 6)*d2Ndxi2dxi4 + aInvJac(:
    end

  otherwise
    vBase = 0;

end
```

11.67 ellib/sf_tri_H3

SF_TRI_H3 Third order 2D C1 Hermite shape functions for triangles.

[VBASE, NLDOF, XLDOF, SFUN] =

SF_TRI_H3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates C1 Hermite shape functions on quadrilaterals with values defined in the nodes, and cell center. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 2              Number of space dimensions
n_vert      scalar: 3              Number of vertices per cell
i_dof       scalar: 1-10           Local basis function to evaluate
xi          array [3,1]            Local coordinates of evaluation point
aInvJac     [n,6]                  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [3,4]                  Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [3,n_ldof]             Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_tri_P1

Code listing

```
nLDof = [3 0 0 1;
         3 0 0 0;
         3 0 0 0];
xLDof = [repmat([1 0 0;
                 0 1 0;
                 0 0 1],1,3) [1/3;1/3;1/3]];
sfun  = 'sf_tri_H3';


switch i_eval

  case 1
```

```
    switch i_dof
      case 1
        vBase = - 2*xi(1)^3 + 3*xi(1)^2 - 7*xi(2)*xi(3)*xi(1);
      case 2
        vBase = - 2*xi(2)^3 + 3*xi(2)^2 - 7*xi(1)*xi(3)*xi(2);
      case 3
        vBase = - 2*xi(3)^3 + 3*xi(3)^2 - 7*xi(1)*xi(2)*xi(3);
      case 4
        vBase = - xi(1)*xi(2)*((-aInvJac(:,6).*aInvJac(:,7)))*(2*xi(1) + xi(2) - 1) - xi(1)*xi(3)
      case 5
        vBase = xi(1)*xi(2)*((-aInvJac(:,6).*aInvJac(:,7)))*(xi(1) + 2*xi(2) - 1) - xi(2)*xi(3)*(
      case 6
        vBase = xi(1)*xi(3)*(aInvJac(:,5).*aInvJac(:,7))*(xi(1) + 2*xi(3) - 1) + xi(2)*xi(3)*((-a
      case 7
        vBase = - xi(1)*xi(2)*(aInvJac(:,3).*aInvJac(:,7))*(2*xi(1) + xi(2) - 1) - xi(1)*xi(3)*((
      case 8
        vBase = xi(1)*xi(2)*(aInvJac(:,3).*aInvJac(:,7))*(xi(1) + 2*xi(2) - 1) - xi(2)*xi(3)*(aIn
      case 9
        vBase = xi(1)*xi(3)*((-aInvJac(:,2).*aInvJac(:,7)))*(xi(1) + 2*xi(3) - 1) + xi(2)*xi(3)*(
      case 10
        vBase = 27*xi(1)*xi(2)*xi(3);
    end

  case {2,3}

    switch i_dof

      case 1
        dNdxi1 = - 6*xi(1)^2 + 6*xi(1) - 7*xi(2)*xi(3);
        dNdxi2 = -7*xi(1)*xi(3);
        dNdxi3 = -7*xi(1)*xi(2);
      case 2
        dNdxi1 = -7*xi(2)*xi(3);
        dNdxi2 = - 6*xi(2)^2 + 6*xi(2) - 7*xi(1)*xi(3);
        dNdxi3 = -7*xi(1)*xi(2);
      case 3
        dNdxi1 = -7*xi(2)*xi(3);
        dNdxi2 = -7*xi(1)*xi(3);
        dNdxi3 = - 6*xi(3)^2 + 6*xi(3) - 7*xi(1)*xi(2);
      case 4
        dNdxi1 = - xi(2)*((-aInvJac(:,6).*aInvJac(:,7)))*(2*xi(1) + xi(2) - 1) - xi(3)*(aInvJac(:
        dNdxi2 = -xi(1)*((-aInvJac(:,6).*aInvJac(:,7)))*(2*xi(1) + 2*xi(2) - 1);
        dNdxi3 = -xi(1)*(aInvJac(:,5).*aInvJac(:,7))*(2*xi(1) + 2*xi(3) - 1);
      case 5
        dNdxi1 = xi(2)*((-aInvJac(:,6).*aInvJac(:,7)))*(2*xi(1) + 2*xi(2) - 1);
        dNdxi2 = xi(1)*((-aInvJac(:,6).*aInvJac(:,7)))*(xi(1) + 2*xi(2) - 1) - xi(3)*((-aInvJac(:
        dNdxi3 = -xi(2)*((-aInvJac(:,4).*aInvJac(:,7)))*(2*xi(2) + 2*xi(3) - 1);
      case 6
        dNdxi1 = xi(3)*(aInvJac(:,5).*aInvJac(:,7))*(2*xi(1) + 2*xi(3) - 1);
        dNdxi2 = xi(3)*((-aInvJac(:,4).*aInvJac(:,7)))*(2*xi(2) + 2*xi(3) - 1);
        dNdxi3 = xi(1)*(aInvJac(:,5).*aInvJac(:,7))*(xi(1) + 2*xi(3) - 1) + xi(2)*((-aInvJac(:,4)
      case 7
```

```
          dNdxi1 = - 2*xi(1)*xi(2)*(aInvJac(:,3).*aInvJac(:,7)) - 2*xi(1)*xi(3)*((-aInvJac(:,2).*aI
          dNdxi2 = -xi(1)*(aInvJac(:,3).*aInvJac(:,7))*(2*xi(1) + 2*xi(2) - 1);
          dNdxi3 = -xi(1)*((-aInvJac(:,2).*aInvJac(:,7)))*(2*xi(1) + 2*xi(3) - 1);
        case 8
          dNdxi1 = xi(2)*(aInvJac(:,3).*aInvJac(:,7))*(2*xi(1) + 2*xi(2) - 1);
          dNdxi2 = 2*xi(1)*xi(2)*(aInvJac(:,3).*aInvJac(:,7)) - 2*xi(2)*xi(3)*(aInvJac(:,1).*aInvJa
          dNdxi3 = -xi(2)*(aInvJac(:,1).*aInvJac(:,7))*(2*xi(2) + 2*xi(3) - 1);
        case 9
          dNdxi1 = xi(3)*((-aInvJac(:,2).*aInvJac(:,7)))*(2*xi(1) + 2*xi(3) - 1);
          dNdxi2 = xi(3)*(aInvJac(:,1).*aInvJac(:,7))*(2*xi(2) + 2*xi(3) - 1);
          dNdxi3 = 2*xi(1)*xi(3)*((-aInvJac(:,2).*aInvJac(:,7))) + 2*xi(2)*xi(3)*(aInvJac(:,1).*aIn
        case 10
          dNdxi1 = 27*xi(2)*xi(3);
          dNdxi2 = 27*xi(1)*xi(3);
          dNdxi3 = 27*xi(1)*xi(2);
      end

      if( i_eval==2 )
        vBase = aInvJac(:,1).*dNdxi1 + aInvJac(:,2).*dNdxi2 + aInvJac(:,3).*dNdxi3;
      else
        vBase = aInvJac(:,4).*dNdxi1 + aInvJac(:,5).*dNdxi2 + aInvJac(:,6).*dNdxi3;
      end

    otherwise
      vBase = 0;

  end
```

11.68 ellib/sf_tri_P3

SF_TRI_P3 Third order Lagrange shape functions for triangles (P3).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_TRI_P3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming third order P3 Lagrange shape functions on 2D triangular elements with values defined in the nodes, edges, and center. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 2              Number of space dimensions
n_vert      scalar: 3              Number of vertices per cell
i_dof       scalar: 1-10           Local basis function to evaluate
xi          array [3,1]            Local coordinates of evaluation point
aInvJac     [n,6]                  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
```

```
Output      Value/[Size]          Description
-------------------------------------------------------------------------------
vBase       [n]                   Evaluated function values
nLDof       [4]                   Number of local degrees of freedom on
                                  vertices, edges, faces, and cell interiors
xLDof       [3,n_ldof]            Local coordinates of local dofs
sfun        string                Function name of called shape function
```

See also

sflag3, sf_tri_H3

Code listing

```
nLDof = [3 6 0 1];
xLDof = [1 0 0 2/3   0 1/3 1/3   0 2/3 1/3;
         0 1 0 1/3 2/3   0 2/3 1/3   0 1/3;
         0 0 1   0 1/3 2/3   0 2/3 1/3 1/3];
sfun  = 'sf_tri_P3';


switch i_eval

  case 1

    switch i_dof
      case 1
        vBase = (xi(1)*(3*xi(1) - 1)*(3*xi(1) - 2))/2;
      case 2
        vBase = (xi(2)*(3*xi(2) - 1)*(3*xi(2) - 2))/2;
      case 3
        vBase = (xi(3)*(3*xi(3) - 1)*(3*xi(3) - 2))/2;
      case 4
        vBase = (9*xi(1)*xi(2)*(3*xi(1) - 1))/2;
      case 5
        vBase = (9*xi(2)*xi(3)*(3*xi(2) - 1))/2;
      case 6
        vBase = (9*xi(1)*xi(3)*(3*xi(3) - 1))/2;
      case 7
        vBase = (9*xi(1)*xi(2)*(3*xi(2) - 1))/2;
      case 8
        vBase = (9*xi(2)*xi(3)*(3*xi(3) - 1))/2;
      case 9
        vBase = (9*xi(1)*xi(3)*(3*xi(1) - 1))/2;
```

```
      case 10
        vBase = 27*xi(1)*xi(2)*xi(3);
    end

  case {2,3}

    switch i_dof

      case 1
        dNdxi1 = (27*xi(1)^2)/2 - 9*xi(1) + 1;
        dNdxi2 = 0;
        dNdxi3 = 0;
      case 2
        dNdxi1 = 0;
        dNdxi2 = (27*xi(2)^2)/2 - 9*xi(2) + 1;
        dNdxi3 = 0;
      case 3
        dNdxi1 = 0;
        dNdxi2 = 0;
        dNdxi3 = (27*xi(3)^2)/2 - 9*xi(3) + 1;
      case 4
        dNdxi1 = (9*xi(2)*(6*xi(1) - 1))/2;
        dNdxi2 = (9*xi(1)*(3*xi(1) - 1))/2;
        dNdxi3 = 0;
      case 5
        dNdxi1 = 0;
        dNdxi2 = (9*xi(3)*(6*xi(2) - 1))/2;
        dNdxi3 = (9*xi(2)*(3*xi(2) - 1))/2;
      case 6
        dNdxi1 = (9*xi(3)*(3*xi(3) - 1))/2;
        dNdxi2 = 0;
        dNdxi3 = (9*xi(1)*(6*xi(3) - 1))/2;
      case 7
        dNdxi1 = (9*xi(2)*(3*xi(2) - 1))/2;
        dNdxi2 = (9*xi(1)*(6*xi(2) - 1))/2;
        dNdxi3 = 0;
      case 8
        dNdxi1 = 0;
        dNdxi2 = (9*xi(3)*(3*xi(3) - 1))/2;
        dNdxi3 = (9*xi(2)*(6*xi(3) - 1))/2;
      case 9
        dNdxi1 = (9*xi(3)*(6*xi(1) - 1))/2;
        dNdxi2 = 0;
        dNdxi3 = (9*xi(1)*(3*xi(1) - 1))/2;
      case 10
        dNdxi1 = 27*xi(2)*xi(3);
        dNdxi2 = 27*xi(1)*xi(3);
        dNdxi3 = 27*xi(1)*xi(2);
    end

    if( i_eval==2 )
```

```
        vBase = aInvJac(:,1)*dNdxi1 + aInvJac(:,2)*dNdxi2 + aInvJac(:,3)*dNdxi3;

      else

        vBase = aInvJac(:,4)*dNdxi1 + aInvJac(:,5)*dNdxi2 + aInvJac(:,6)*dNdxi3;

      end

    case {22,23,32,33}   % Evaluation of second derivatives.

      switch i_dof

        case 1
          d2Ndxi1dxi1 = 27*xi(1) - 9;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 2
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 27*xi(2) - 9;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 3
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 27*xi(3) - 9;

        case 4
          d2Ndxi1dxi1 = 27*xi(2);
          d2Ndxi2dxi1 = 27*xi(1) - 9/2;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 27*xi(1) - 9/2;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
```

```
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 5
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 27*xi(3);
          d2Ndxi3dxi2 = 27*xi(2) - 9/2;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 27*xi(2) - 9/2;
          d2Ndxi3dxi3 = 0;

        case 6
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 27*xi(3) - 9/2;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 27*xi(3) - 9/2;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 27*xi(1);

        case 7
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 27*xi(2) - 9/2;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 27*xi(2) - 9/2;
          d2Ndxi2dxi2 = 27*xi(1);
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 8
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 27*xi(3) - 9/2;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 27*xi(3) - 9/2;
          d2Ndxi3dxi3 = 27*xi(2);

        case 9
          d2Ndxi1dxi1 = 27*xi(3);
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 27*xi(1) - 9/2;
```

```
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 27*xi(1) - 9/2;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 10
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 27*xi(3);
          d2Ndxi3dxi1 = 27*xi(2);
          d2Ndxi1dxi2 = 27*xi(3);
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 27*xi(1);
          d2Ndxi1dxi3 = 27*xi(2);
          d2Ndxi2dxi3 = 27*xi(1);
          d2Ndxi3dxi3 = 0;
      end

      if( i_eval==22 )
        vBase = aInvJac(:,1).*( aInvJac(:,1).*d2Ndxi1dxi1 + aInvJac(:,2).*d2Ndxi2dxi1 + aInvJac(:,3
                aInvJac(:,2).*( aInvJac(:,1).*d2Ndxi1dxi2 + aInvJac(:,2).*d2Ndxi2dxi2 + aInvJac(:,3
                aInvJac(:,3).*( aInvJac(:,1).*d2Ndxi1dxi3 + aInvJac(:,2).*d2Ndxi2dxi3 + aInvJac(:,3
      elseif( i_eval==33 )
        vBase = aInvJac(:,4).*( aInvJac(:,4).*d2Ndxi1dxi1 + aInvJac(:,5).*d2Ndxi2dxi1 + aInvJac(:,6
                aInvJac(:,5).*( aInvJac(:,4).*d2Ndxi1dxi2 + aInvJac(:,5).*d2Ndxi2dxi2 + aInvJac(:,6
                aInvJac(:,6).*( aInvJac(:,4).*d2Ndxi1dxi3 + aInvJac(:,5).*d2Ndxi2dxi3 + aInvJac(:,6
      else
        vBase = aInvJac(:,4).*( aInvJac(:,1).*d2Ndxi1dxi1 + aInvJac(:,2).*d2Ndxi2dxi1 + aInvJac(:,3
                aInvJac(:,5).*( aInvJac(:,1).*d2Ndxi1dxi2 + aInvJac(:,2).*d2Ndxi2dxi2 + aInvJac(:,3
                aInvJac(:,6).*( aInvJac(:,1).*d2Ndxi1dxi3 + aInvJac(:,2).*d2Ndxi2dxi3 + aInvJac(:,3
      end

    otherwise
      vBase = 0;

  end
```

11.69 ellib/sf_tri_P4

SF_TRI_P4 Fourth order Lagrange shape functions for triangles (P4).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_TRI_P4(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming fourth order P4 Lagrange shape functions on 2D triangular elements with values defined in the nodes, edges, and center. XI are Barycentric coordinates.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 2              Number of space dimensions
n_vert      scalar: 3              Number of vertices per cell
i_dof       scalar: 1-15           Local basis function to evaluate
xi          array [3,1]            Local coordinates of evaluation point
aInvJac     [n,6]                  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [3,n_ldof]             Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_tri_P1

Code listing

```
nLDof = [3 9 0 3];
xLDof = [1 0 0 3/4   0 1/4 1/2   0 1/2 1/4   0 3/4 1/2 1/4 1/4;
         0 1 0 1/4 3/4   0 1/2 1/2   0 3/4 1/4   0 1/4 1/2 1/4;
         0 0 1   0 1/4 3/4   0 1/2 1/2   0 3/4 1/4 1/4 1/4 1/2];
sfun  = 'sf_tri_P4';


switch i_eval

  case 1

    switch i_dof
      case 1
        vBase = (xi(1)*(2*xi(1) - 1)*(4*xi(1) - 1)*(4*xi(1) - 3))/3;
      case 2
        vBase = (xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1)*(4*xi(2) - 3))/3;
      case 3
        vBase = (xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1)*(4*xi(3) - 3))/3;
      case 4
        vBase = (16*xi(1)*xi(2)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
```

```
      case 5
        vBase = (16*xi(2)*xi(3)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
      case 6
        vBase = (16*xi(1)*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
      case 7
        vBase = 4*xi(1)*xi(2)*(4*xi(1) - 1)*(4*xi(2) - 1);
      case 8
        vBase = 4*xi(2)*xi(3)*(4*xi(2) - 1)*(4*xi(3) - 1);
      case 9
        vBase = 4*xi(1)*xi(3)*(4*xi(1) - 1)*(4*xi(3) - 1);
      case 10
        vBase = (16*xi(1)*xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
      case 11
        vBase = (16*xi(2)*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
      case 12
        vBase = (16*xi(1)*xi(3)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
      case 13
        vBase = 32*xi(1)*xi(2)*xi(3)*(4*xi(1) - 1);
      case 14
        vBase = 32*xi(1)*xi(2)*xi(3)*(4*xi(2) - 1);
      case 15
        vBase = 32*xi(1)*xi(2)*xi(3)*(4*xi(3) - 1);
    end

  case {2,3}

    switch i_dof

      case 1
        dNdxi1 = ((8*xi(1) - 3)*(16*xi(1)^2 - 12*xi(1) + 1))/3;
        dNdxi2 = 0;
        dNdxi3 = 0;
      case 2
        dNdxi1 = 0;
        dNdxi2 = ((8*xi(2) - 3)*(16*xi(2)^2 - 12*xi(2) + 1))/3;
        dNdxi3 = 0;
      case 3
        dNdxi1 = 0;
        dNdxi2 = 0;
        dNdxi3 = ((8*xi(3) - 3)*(16*xi(3)^2 - 12*xi(3) + 1))/3;
      case 4
        dNdxi1 = (16*xi(2)*(24*xi(1)^2 - 12*xi(1) + 1))/3;
        dNdxi2 = (16*xi(1)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
        dNdxi3 = 0;
      case 5
        dNdxi1 = 0;
        dNdxi2 = (16*xi(3)*(24*xi(2)^2 - 12*xi(2) + 1))/3;
        dNdxi3 = (16*xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
      case 6
        dNdxi1 = (16*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
        dNdxi2 = 0;
        dNdxi3 = (16*xi(1)*(24*xi(3)^2 - 12*xi(3) + 1))/3;
```

```
        case 7
          dNdxi1 = 4*xi(2)*(8*xi(1) - 1)*(4*xi(2) - 1);
          dNdxi2 = 4*xi(1)*(4*xi(1) - 1)*(8*xi(2) - 1);
          dNdxi3 = 0;
        case 8
          dNdxi1 = 0;
          dNdxi2 = 4*xi(3)*(8*xi(2) - 1)*(4*xi(3) - 1);
          dNdxi3 = 4*xi(2)*(4*xi(2) - 1)*(8*xi(3) - 1);
        case 9
          dNdxi1 = 4*xi(3)*(8*xi(1) - 1)*(4*xi(3) - 1);
          dNdxi2 = 0;
          dNdxi3 = 4*xi(1)*(4*xi(1) - 1)*(8*xi(3) - 1);
        case 10
          dNdxi1 = (16*xi(2)*(2*xi(2) - 1)*(4*xi(2) - 1))/3;
          dNdxi2 = (16*xi(1)*(24*xi(2)^2 - 12*xi(2) + 1))/3;
          dNdxi3 = 0;
        case 11
          dNdxi1 = 0;
          dNdxi2 = (16*xi(3)*(2*xi(3) - 1)*(4*xi(3) - 1))/3;
          dNdxi3 = (16*xi(2)*(24*xi(3)^2 - 12*xi(3) + 1))/3;
        case 12
          dNdxi1 = (16*xi(3)*(24*xi(1)^2 - 12*xi(1) + 1))/3;
          dNdxi2 = 0;
          dNdxi3 = (16*xi(1)*(2*xi(1) - 1)*(4*xi(1) - 1))/3;
        case 13
          dNdxi1 = 32*xi(2)*xi(3)*(8*xi(1) - 1);
          dNdxi2 = 32*xi(1)*xi(3)*(4*xi(1) - 1);
          dNdxi3 = 32*xi(1)*xi(2)*(4*xi(1) - 1);
        case 14
          dNdxi1 = 32*xi(2)*xi(3)*(4*xi(2) - 1);
          dNdxi2 = 32*xi(1)*xi(3)*(8*xi(2) - 1);
          dNdxi3 = 32*xi(1)*xi(2)*(4*xi(2) - 1);
        case 15
          dNdxi1 = 32*xi(2)*xi(3)*(4*xi(3) - 1);
          dNdxi2 = 32*xi(1)*xi(3)*(4*xi(3) - 1);
          dNdxi3 = 32*xi(1)*xi(2)*(8*xi(3) - 1);
      end

      if( i_eval==2 )

        vBase = aInvJac(:,1).*dNdxi1 + aInvJac(:,2).*dNdxi2 + aInvJac(:,3).*dNdxi3;

      else

        vBase = aInvJac(:,4).*dNdxi1 + aInvJac(:,5).*dNdxi2 + aInvJac(:,6).*dNdxi3;

      end

    case {22,23,32,33}   % Evaluation of second derivatives.

      switch i_dof
```

```
        case 1
          d2Ndxi1dxi1 = 128*xi(1)^2 - 96*xi(1) + 44/3;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 2
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 128*xi(2)^2 - 96*xi(2) + 44/3;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 3
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 128*xi(3)^2 - 96*xi(3) + 44/3;

        case 4
          d2Ndxi1dxi1 = 64*xi(2)*(4*xi(1) - 1);
          d2Ndxi2dxi1 = 128*xi(1)^2 - 64*xi(1) + 16/3;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 128*xi(1)^2 - 64*xi(1) + 16/3;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 5
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 64*xi(3)*(4*xi(2) - 1);
          d2Ndxi3dxi2 = 128*xi(2)^2 - 64*xi(2) + 16/3;
          d2Ndxi1dxi3 = 0;
```

```
    d2Ndxi2dxi3 = 128*xi(2)^2 - 64*xi(2) + 16/3;
    d2Ndxi3dxi3 = 0;

  case 6
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 128*xi(3)^2 - 64*xi(3) + 16/3;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 128*xi(3)^2 - 64*xi(3) + 16/3;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 64*xi(1)*(4*xi(3) - 1);

  case 7
    d2Ndxi1dxi1 = 32*xi(2)*(4*xi(2) - 1);
    d2Ndxi2dxi1 = 4*(8*xi(1) - 1)*(8*xi(2) - 1);
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = 4*(8*xi(1) - 1)*(8*xi(2) - 1);
    d2Ndxi2dxi2 = 32*xi(1)*(4*xi(1) - 1);
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;

  case 8
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 32*xi(3)*(4*xi(3) - 1);
    d2Ndxi3dxi2 = 4*(8*xi(2) - 1)*(8*xi(3) - 1);
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 4*(8*xi(2) - 1)*(8*xi(3) - 1);
    d2Ndxi3dxi3 = 32*xi(2)*(4*xi(2) - 1);

  case 9
    d2Ndxi1dxi1 = 32*xi(3)*(4*xi(3) - 1);
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 4*(8*xi(1) - 1)*(8*xi(3) - 1);
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 4*(8*xi(1) - 1)*(8*xi(3) - 1);
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 32*xi(1)*(4*xi(1) - 1);

  case 10
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 128*xi(2)^2 - 64*xi(2) + 16/3;
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = 128*xi(2)^2 - 64*xi(2) + 16/3;
```

```
    d2Ndxi2dxi2 = 64*xi(1)*(4*xi(2) - 1);
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;

  case 11
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 128*xi(3)^2 - 64*xi(3) + 16/3;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 128*xi(3)^2 - 64*xi(3) + 16/3;
    d2Ndxi3dxi3 = 64*xi(2)*(4*xi(3) - 1);

  case 12
    d2Ndxi1dxi1 = 64*xi(3)*(4*xi(1) - 1);
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 128*xi(1)^2 - 64*xi(1) + 16/3;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 128*xi(1)^2 - 64*xi(1) + 16/3;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;

  case 13
    d2Ndxi1dxi1 = 256*xi(2)*xi(3);
    d2Ndxi2dxi1 = 32*xi(3)*(8*xi(1) - 1);
    d2Ndxi3dxi1 = 32*xi(2)*(8*xi(1) - 1);
    d2Ndxi1dxi2 = 32*xi(3)*(8*xi(1) - 1);
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 32*xi(1)*(4*xi(1) - 1);
    d2Ndxi1dxi3 = 32*xi(2)*(8*xi(1) - 1);
    d2Ndxi2dxi3 = 32*xi(1)*(4*xi(1) - 1);
    d2Ndxi3dxi3 = 0;

  case 14
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 32*xi(3)*(8*xi(2) - 1);
    d2Ndxi3dxi1 = 32*xi(2)*(4*xi(2) - 1);
    d2Ndxi1dxi2 = 32*xi(3)*(8*xi(2) - 1);
    d2Ndxi2dxi2 = 256*xi(1)*xi(3);
    d2Ndxi3dxi2 = 32*xi(1)*(8*xi(2) - 1);
    d2Ndxi1dxi3 = 32*xi(2)*(4*xi(2) - 1);
    d2Ndxi2dxi3 = 32*xi(1)*(8*xi(2) - 1);
    d2Ndxi3dxi3 = 0;

  case 15
    d2Ndxi1dxi1 = 0;
```

```
          d2Ndxi2dxi1 = 32*xi(3)*(4*xi(3) - 1);
          d2Ndxi3dxi1 = 32*xi(2)*(8*xi(3) - 1);
          d2Ndxi1dxi2 = 32*xi(3)*(4*xi(3) - 1);
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 32*xi(1)*(8*xi(3) - 1);
          d2Ndxi1dxi3 = 32*xi(2)*(8*xi(3) - 1);
          d2Ndxi2dxi3 = 32*xi(1)*(8*xi(3) - 1);
          d2Ndxi3dxi3 = 256*xi(1)*xi(2);
      end

      if( i_eval==22 )
        vBase = aInvJac(:,1).*( aInvJac(:,1).*d2Ndxi1dxi1 + aInvJac(:,2).*d2Ndxi2dxi1 + aInvJac(:,3
                aInvJac(:,2).*( aInvJac(:,1).*d2Ndxi1dxi2 + aInvJac(:,2).*d2Ndxi2dxi2 + aInvJac(:,3
                aInvJac(:,3).*( aInvJac(:,1).*d2Ndxi1dxi3 + aInvJac(:,2).*d2Ndxi2dxi3 + aInvJac(:,3
      elseif( i_eval==33 )
        vBase = aInvJac(:,4).*( aInvJac(:,4).*d2Ndxi1dxi1 + aInvJac(:,5).*d2Ndxi2dxi1 + aInvJac(:,6
                aInvJac(:,5).*( aInvJac(:,4).*d2Ndxi1dxi2 + aInvJac(:,5).*d2Ndxi2dxi2 + aInvJac(:,6
                aInvJac(:,6).*( aInvJac(:,4).*d2Ndxi1dxi3 + aInvJac(:,5).*d2Ndxi2dxi3 + aInvJac(:,6
      else
        vBase = aInvJac(:,4).*( aInvJac(:,1).*d2Ndxi1dxi1 + aInvJac(:,2).*d2Ndxi2dxi1 + aInvJac(:,3
                aInvJac(:,5).*( aInvJac(:,1).*d2Ndxi1dxi2 + aInvJac(:,2).*d2Ndxi2dxi2 + aInvJac(:,3
                aInvJac(:,6).*( aInvJac(:,1).*d2Ndxi1dxi3 + aInvJac(:,2).*d2Ndxi2dxi3 + aInvJac(:,3
      end

    otherwise
      vBase = 0;

  end
```

11.70 ellib/sf_tri_P5

SF_TRI_P5 Fifth order Lagrange shape functions for triangles (P5).

[VBASE, NLDOF, XLDOF, SFUN] =

SF_TRI_P5(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming fifth order P5 Lagrange shape functions on 2D triangular elements with values defined in the nodes, edges, and center. XI are Barycentric coordinates.

```
Input       Value/[Size]          Description
-----------------------------------------------------------------------------
i_eval      scalar:  1            Evaluate function values
                    >1            Evaluate values of derivatives
n_sdim      scalar: 2             Number of space dimensions
n_vert      scalar: 3             Number of vertices per cell
i_dof       scalar: 1-21          Local basis function to evaluate
xi          array [3,1]           Local coordinates of evaluation point
aInvJac     [n,6]                 Inverse of transformation Jacobian
```

```
vBase       [n]                   Preallocated output vector
                                                                                .
Output      Value/[Size]          Description
--------------------------------------------------------------------------------
vBase       [n]                   Evaluated function values
nLDof       [4]                   Number of local degrees of freedom on
                                  vertices, edges, faces, and cell interiors
xLDof       [3,n_ldof]            Local coordinates of local dofs
sfun        string                Function name of called shape function
```

See also

sf_tri_P1

Code listing

```
nLDof = [3 12 0 6];
xLDof = [1 0 0 4/5   0 1/5 3/5   0 2/5 2/5   0 3/5 1/5   0 4/5 3/5 1/5 1/5 2/5 1/5 2/5;
         0 1 0 1/5 4/5   0 2/5 3/5   0 3/5 2/5   0 4/5 1/5   0 1/5 3/5 1/5 2/5 2/5 1/5;
         0 0 1   0 1/5 4/5   0 2/5 3/5   0 3/5 2/5   0 4/5 1/5 1/5 1/5 3/5 1/5 2/5 2/5];
sfun  = 'sf_tri_P5';


switch i_eval

  case 1

    switch i_dof
      case 1
        vBase = (xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3)*(5*xi(1) - 4))/24;
      case 2
        vBase = (xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3)*(5*xi(2) - 4))/24;
      case 3
        vBase = (xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3)*(5*xi(3) - 4))/24;
      case 4
        vBase = (25*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
      case 5
        vBase = (25*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
      case 6
        vBase = (25*xi(1)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
      case 7
        vBase = (25*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(2) - 1))/12;
      case 8
        vBase = (25*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(3) - 1))/12;
      case 9
```

```
          vBase = (25*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1)*(5*xi(3) - 2))/12;
        case 10
          vBase = (25*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(2) - 1)*(5*xi(2) - 2))/12;
        case 11
          vBase = (25*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1)*(5*xi(3) - 2))/12;
        case 12
          vBase = (25*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(3) - 1))/12;
        case 13
          vBase = (25*xi(1)*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
        case 14
          vBase = (25*xi(2)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
        case 15
          vBase = (25*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
        case 16
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
        case 17
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
        case 18
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
        case 19
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(1) - 1)*(5*xi(2) - 1))/4;
        case 20
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1))/4;
        case 21
          vBase = (125*xi(1)*xi(2)*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1))/4;
      end

    case {2,3}

      switch i_dof

        case 1
          dNdxi1 = (3125*xi(1)^4)/24 - (625*xi(1)^3)/3 + (875*xi(1)^2)/8 - (125*xi(1))/6 + 1;
          dNdxi2 = 0;
          dNdxi3 = 0;
        case 2
          dNdxi1 = 0;
          dNdxi2 = (3125*xi(2)^4)/24 - (625*xi(2)^3)/3 + (875*xi(2)^2)/8 - (125*xi(2))/6 + 1;
          dNdxi3 = 0;
        case 3
          dNdxi1 = 0;
          dNdxi2 = 0;
          dNdxi3 = (3125*xi(3)^4)/24 - (625*xi(3)^3)/3 + (875*xi(3)^2)/8 - (125*xi(3))/6 + 1;
        case 4
          dNdxi1 = (25*xi(2)*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
          dNdxi2 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
          dNdxi3 = 0;
        case 5
          dNdxi1 = 0;
          dNdxi2 = (25*xi(3)*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
          dNdxi3 = (25*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
        case 6
```

```
    dNdxi1 = (25*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
    dNdxi2 = 0;
    dNdxi3 = (25*xi(1)*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
  case 7
    dNdxi1 = (25*xi(2)*(5*xi(2) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
    dNdxi2 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(10*xi(2) - 1))/12;
    dNdxi3 = 0;
  case 8
    dNdxi1 = 0;
    dNdxi2 = (25*xi(3)*(5*xi(3) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
    dNdxi3 = (25*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(10*xi(3) - 1))/12;
  case 9
    dNdxi1 = (25*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1)*(5*xi(3) - 2))/12;
    dNdxi2 = 0;
    dNdxi3 = (25*xi(1)*(5*xi(1) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
  case 10
    dNdxi1 = (25*xi(2)*(10*xi(1) - 1)*(5*xi(2) - 1)*(5*xi(2) - 2))/12;
    dNdxi2 = (25*xi(1)*(5*xi(1) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
    dNdxi3 = 0;
  case 11
    dNdxi1 = 0;
    dNdxi2 = (25*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1)*(5*xi(3) - 2))/12;
    dNdxi3 = (25*xi(2)*(5*xi(2) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
  case 12
    dNdxi1 = (25*xi(3)*(5*xi(3) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
    dNdxi2 = 0;
    dNdxi3 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(10*xi(3) - 1))/12;
  case 13
    dNdxi1 = (25*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2)*(5*xi(2) - 3))/24;
    dNdxi2 = (25*xi(1)*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
    dNdxi3 = 0;
  case 14
    dNdxi1 = 0;
    dNdxi2 = (25*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2)*(5*xi(3) - 3))/24;
    dNdxi3 = (25*xi(2)*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
  case 15
    dNdxi1 = (25*xi(3)*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
    dNdxi2 = 0;
    dNdxi3 = (25*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2)*(5*xi(1) - 3))/24;
  case 16
    dNdxi1 = (125*xi(2)*xi(3)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    dNdxi2 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    dNdxi3 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
  case 17
    dNdxi1 = (125*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
    dNdxi2 = (125*xi(1)*xi(3)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    dNdxi3 = (125*xi(1)*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
  case 18
    dNdxi1 = (125*xi(2)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
    dNdxi2 = (125*xi(1)*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
    dNdxi3 = (125*xi(1)*xi(2)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
  case 19
```

```
        dNdxi1 = (125*xi(2)*xi(3)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
        dNdxi2 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
        dNdxi3 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(5*xi(2) - 1))/4;
      case 20
        dNdxi1 = (125*xi(2)*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1))/4;
        dNdxi2 = (125*xi(1)*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
        dNdxi3 = (125*xi(1)*xi(2)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
      case 21
        dNdxi1 = (125*xi(2)*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
        dNdxi2 = (125*xi(1)*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1))/4;
        dNdxi3 = (125*xi(1)*xi(2)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
    end

    if( i_eval==2 )

      vBase = aInvJac(:,1).*dNdxi1 + aInvJac(:,2).*dNdxi2 + aInvJac(:,3).*dNdxi3;

    else

      vBase = aInvJac(:,4).*dNdxi1 + aInvJac(:,5).*dNdxi2 + aInvJac(:,6).*dNdxi3;

    end

  case {22,23,32,33}   % Evaluation of second derivatives.

    switch i_dof

      case 1
        d2Ndxi1dxi1 = (125*(5*xi(1) - 2)*(10*xi(1)^2 - 8*xi(1) + 1))/12;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = 0;
        d2Ndxi3dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;

      case 2
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
        d2Ndxi3dxi1 = 0;
        d2Ndxi1dxi2 = 0;
        d2Ndxi2dxi2 = (125*(5*xi(2) - 2)*(10*xi(2)^2 - 8*xi(2) + 1))/12;
        d2Ndxi3dxi2 = 0;
        d2Ndxi1dxi3 = 0;
        d2Ndxi2dxi3 = 0;
        d2Ndxi3dxi3 = 0;

      case 3
        d2Ndxi1dxi1 = 0;
        d2Ndxi2dxi1 = 0;
```

```
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = (125*(5*xi(3) - 2)*(10*xi(3)^2 - 8*xi(3) + 1))/12;

  case 4
    d2Ndxi1dxi1 = (125*xi(2)*(150*xi(1)^2 - 90*xi(1) + 11))/12;
    d2Ndxi2dxi1 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;

  case 5
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = (125*xi(3)*(150*xi(2)^2 - 90*xi(2) + 11))/12;
    d2Ndxi3dxi2 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
    d2Ndxi3dxi3 = 0;

  case 6
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = (125*xi(1)*(150*xi(3)^2 - 90*xi(3) + 11))/12;

  case 7
    d2Ndxi1dxi1 = (125*xi(2)*(5*xi(1) - 1)*(5*xi(2) - 1))/2;
    d2Ndxi2dxi1 = (25*(10*xi(2) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = (25*(10*xi(2) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
    d2Ndxi2dxi2 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;
```

```
        case 8
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = (125*xi(3)*(5*xi(2) - 1)*(5*xi(3) - 1))/2;
          d2Ndxi3dxi2 = (25*(10*xi(3) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = (25*(10*xi(3) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
          d2Ndxi3dxi3 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;

        case 9
          d2Ndxi1dxi1 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = (25*(10*xi(1) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = (25*(10*xi(1) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(3) - 1))/2;

        case 10
          d2Ndxi1dxi1 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
          d2Ndxi2dxi1 = (25*(10*xi(1) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = (25*(10*xi(1) - 1)*(75*xi(2)^2 - 30*xi(2) + 2))/12;
          d2Ndxi2dxi2 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(2) - 1))/2;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = 0;
          d2Ndxi3dxi3 = 0;

        case 11
          d2Ndxi1dxi1 = 0;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = 0;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
          d2Ndxi3dxi2 = (25*(10*xi(2) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
          d2Ndxi1dxi3 = 0;
          d2Ndxi2dxi3 = (25*(10*xi(2) - 1)*(75*xi(3)^2 - 30*xi(3) + 2))/12;
          d2Ndxi3dxi3 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(3) - 1))/2;

        case 12
          d2Ndxi1dxi1 = (125*xi(3)*(5*xi(1) - 1)*(5*xi(3) - 1))/2;
          d2Ndxi2dxi1 = 0;
          d2Ndxi3dxi1 = (25*(10*xi(3) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
          d2Ndxi1dxi2 = 0;
          d2Ndxi2dxi2 = 0;
          d2Ndxi3dxi2 = 0;
          d2Ndxi1dxi3 = (25*(10*xi(3) - 1)*(75*xi(1)^2 - 30*xi(1) + 2))/12;
```

```
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;

  case 13
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = (25*(10*xi(2) - 3)*(25*xi(2)^2 - 15*xi(2) + 1))/12;
    d2Ndxi2dxi2 = (125*xi(1)*(150*xi(2)^2 - 90*xi(2) + 11))/12;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;

  case 14
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = 0;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
    d2Ndxi1dxi3 = 0;
    d2Ndxi2dxi3 = (25*(10*xi(3) - 3)*(25*xi(3)^2 - 15*xi(3) + 1))/12;
    d2Ndxi3dxi3 = (125*xi(2)*(150*xi(3)^2 - 90*xi(3) + 11))/12;

  case 15
    d2Ndxi1dxi1 = (125*xi(3)*(150*xi(1)^2 - 90*xi(1) + 11))/12;
    d2Ndxi2dxi1 = 0;
    d2Ndxi3dxi1 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
    d2Ndxi1dxi2 = 0;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = 0;
    d2Ndxi1dxi3 = (25*(10*xi(1) - 3)*(25*xi(1)^2 - 15*xi(1) + 1))/12;
    d2Ndxi2dxi3 = 0;
    d2Ndxi3dxi3 = 0;

  case 16
    d2Ndxi1dxi1 = 625*xi(2)*xi(3)*(5*xi(1) - 1);
    d2Ndxi2dxi1 = (125*xi(3)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi3dxi1 = (125*xi(2)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi1dxi2 = (125*xi(3)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi2dxi2 = 0;
    d2Ndxi3dxi2 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi1dxi3 = (125*xi(2)*(75*xi(1)^2 - 30*xi(1) + 2))/6;
    d2Ndxi2dxi3 = (125*xi(1)*(5*xi(1) - 1)*(5*xi(1) - 2))/6;
    d2Ndxi3dxi3 = 0;

  case 17
    d2Ndxi1dxi1 = 0;
    d2Ndxi2dxi1 = (125*xi(3)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
    d2Ndxi3dxi1 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
    d2Ndxi1dxi2 = (125*xi(3)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
```

```
      d2Ndxi2dxi2 = 625*xi(1)*xi(3)*(5*xi(2) - 1);
      d2Ndxi3dxi2 = (125*xi(1)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
      d2Ndxi1dxi3 = (125*xi(2)*(5*xi(2) - 1)*(5*xi(2) - 2))/6;
      d2Ndxi2dxi3 = (125*xi(1)*(75*xi(2)^2 - 30*xi(2) + 2))/6;
      d2Ndxi3dxi3 = 0;

    case 18
      d2Ndxi1dxi1 = 0;
      d2Ndxi2dxi1 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
      d2Ndxi3dxi1 = (125*xi(2)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
      d2Ndxi1dxi2 = (125*xi(3)*(5*xi(3) - 1)*(5*xi(3) - 2))/6;
      d2Ndxi2dxi2 = 0;
      d2Ndxi3dxi2 = (125*xi(1)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
      d2Ndxi1dxi3 = (125*xi(2)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
      d2Ndxi2dxi3 = (125*xi(1)*(75*xi(3)^2 - 30*xi(3) + 2))/6;
      d2Ndxi3dxi3 = 625*xi(1)*xi(2)*(5*xi(3) - 1);

    case 19
      d2Ndxi1dxi1 = (625*xi(2)*xi(3)*(5*xi(2) - 1))/2;
      d2Ndxi2dxi1 = (125*xi(3)*(10*xi(1) - 1)*(10*xi(2) - 1))/4;
      d2Ndxi3dxi1 = (125*xi(2)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
      d2Ndxi1dxi2 = (125*xi(3)*(10*xi(1) - 1)*(10*xi(2) - 1))/4;
      d2Ndxi2dxi2 = (625*xi(1)*xi(3)*(5*xi(1) - 1))/2;
      d2Ndxi3dxi2 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
      d2Ndxi1dxi3 = (125*xi(2)*(10*xi(1) - 1)*(5*xi(2) - 1))/4;
      d2Ndxi2dxi3 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(2) - 1))/4;
      d2Ndxi3dxi3 = 0;

    case 20
      d2Ndxi1dxi1 = 0;
      d2Ndxi2dxi1 = (125*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
      d2Ndxi3dxi1 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
      d2Ndxi1dxi2 = (125*xi(3)*(10*xi(2) - 1)*(5*xi(3) - 1))/4;
      d2Ndxi2dxi2 = (625*xi(1)*xi(3)*(5*xi(3) - 1))/2;
      d2Ndxi3dxi2 = (125*xi(1)*(10*xi(2) - 1)*(10*xi(3) - 1))/4;
      d2Ndxi1dxi3 = (125*xi(2)*(5*xi(2) - 1)*(10*xi(3) - 1))/4;
      d2Ndxi2dxi3 = (125*xi(1)*(10*xi(2) - 1)*(10*xi(3) - 1))/4;
      d2Ndxi3dxi3 = (625*xi(1)*xi(2)*(5*xi(2) - 1))/2;

    case 21
      d2Ndxi1dxi1 = (625*xi(2)*xi(3)*(5*xi(3) - 1))/2;
      d2Ndxi2dxi1 = (125*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
      d2Ndxi3dxi1 = (125*xi(2)*(10*xi(1) - 1)*(10*xi(3) - 1))/4;
      d2Ndxi1dxi2 = (125*xi(3)*(10*xi(1) - 1)*(5*xi(3) - 1))/4;
      d2Ndxi2dxi2 = 0;
      d2Ndxi3dxi2 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
      d2Ndxi1dxi3 = (125*xi(2)*(10*xi(1) - 1)*(10*xi(3) - 1))/4;
      d2Ndxi2dxi3 = (125*xi(1)*(5*xi(1) - 1)*(10*xi(3) - 1))/4;
      d2Ndxi3dxi3 = (625*xi(1)*xi(2)*(5*xi(1) - 1))/2;
  end

  if( i_eval==22 )
```

```
        vBase = aInvJac(:,1).*( aInvJac(:,1).*d2Ndxi1dxi1 + aInvJac(:,2).*d2Ndxi2dxi1 + aInvJac(:,3
                aInvJac(:,2).*( aInvJac(:,1).*d2Ndxi1dxi2 + aInvJac(:,2).*d2Ndxi2dxi2 + aInvJac(:,3
                aInvJac(:,3).*( aInvJac(:,1).*d2Ndxi1dxi3 + aInvJac(:,2).*d2Ndxi2dxi3 + aInvJac(:,3
      elseif( i_eval==33 )
        vBase = aInvJac(:,4).*( aInvJac(:,4).*d2Ndxi1dxi1 + aInvJac(:,5).*d2Ndxi2dxi1 + aInvJac(:,6
                aInvJac(:,5).*( aInvJac(:,4).*d2Ndxi1dxi2 + aInvJac(:,5).*d2Ndxi2dxi2 + aInvJac(:,6
                aInvJac(:,6).*( aInvJac(:,4).*d2Ndxi1dxi3 + aInvJac(:,5).*d2Ndxi2dxi3 + aInvJac(:,6
      else
        vBase = aInvJac(:,4).*( aInvJac(:,1).*d2Ndxi1dxi1 + aInvJac(:,2).*d2Ndxi2dxi1 + aInvJac(:,3
                aInvJac(:,5).*( aInvJac(:,1).*d2Ndxi1dxi2 + aInvJac(:,2).*d2Ndxi2dxi2 + aInvJac(:,3
                aInvJac(:,6).*( aInvJac(:,1).*d2Ndxi1dxi3 + aInvJac(:,2).*d2Ndxi2dxi3 + aInvJac(:,3
      end

    otherwise
      vBase = 0;

  end
```

11.71 ellib/sflag1

SFLAG1 Shape function driver (first order P1/Q1 Lagrange polynomials).

[VBASE, NLDOF, XLDOF, SFUN] =

SFLAG1(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming linear P1 Lagrange shape functions with values defined in the nodes.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_simp_P1, sf_quad_Q3, sf_hex_Q3

Code listing

```
if ( (n_vert==n_sdim+1) || (n_sdim==1) )   % Simplex cells (1D line, 2D triangle, 3D tetrahedron)
  [vBase,nLDof,xLDof,sfun] = sf_simp_P1(i_eval,n_sdim,n_vert,varargin{:});
else   % Quadrilateral and hexahedral cells.
  switch n_sdim
    case 1
      error('sflag1: 1D only supports barycentric coordinates.')
    case 2   % 2D Quadrilateral cells.
      [vBase,nLDof,xLDof,sfun] = sf_quad_Q1(i_eval,n_sdim,n_vert,varargin{:});
    case 3   % 3D Hedahedral cells.
      [vBase,nLDof,xLDof,sfun] = sf_hex_Q1(i_eval,n_sdim,n_vert,varargin{:});
  end
end
```

11.72 ellib/sflag2

SFLAG2 Shape function driver (second order P2/Q2 Lagrange polynomials).

[VBASE, NLDOF, XLDOF, SFUN] =

SFLAG2(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming quadratic Lagrange shape functions with values defined in the nodes, and edges, (and also faces, and cell centers for quadrilaterals and hexahedrals).

```
Input       Value/[Size]           Description
--------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
```

```
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
--------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_simp_P2, sf_quad_Q3, sf_hex_Q3

Code listing

```
if ( (n_vert==n_sdim+1) || (n_sdim==1) )   % Simplex cells (1D line, 2D triangle, 3D tetrahedron)

    [vBase,nLDof,xLDof,sfun] = sf_simp_P2(i_eval,n_sdim,n_vert,varargin{:});

else

  switch n_sdim

    case 2   % 2D Quadrilateral cells.

      [vBase,nLDof,xLDof,sfun] = sf_quad_Q2(i_eval,n_sdim,n_vert,varargin{:});

    case 3   % 3D Hedahedral cells.

      [vBase,nLDof,xLDof,sfun] = sf_hex_Q2(i_eval,n_sdim,n_vert,varargin{:});

  end
end
```

11.73 ellib/sflag3

SFLAG3 Shape function driver (third order P3/Q3 Lagrange polynomials).

[VBASE, NLDOF, XLDOF, SFUN] =

SFLAG3(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming cubic Lagrange shape functions with values defined in the nodes, and edges, (and also faces, and cell centers for quadrilaterals, tetrahedra, and hexahedrals).

```
Input       Value/[Size]           Description
----------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                          .
Output      Value/[Size]           Description
----------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_line_P3, sf_tri_P3, sf_tet_P3, sf_quad_Q3, sf_hex_Q3

Code listing

```
if ( (n_vert==n_sdim+1) || (n_sdim==1) )   % Simplex cells (1D line, 2D triangle, 3D tetrahedron)

  switch n_sdim

    case 1   % 1D Line cells.

      [vBase,nLDof,xLDof,sfun] = sf_line_P3(i_eval,n_sdim,n_vert,varargin{:});

    case 2   % 2D Triangular cells.

      [vBase,nLDof,xLDof,sfun] = sf_tri_P3(i_eval,n_sdim,n_vert,varargin{:});

    case 3   % 3D Tetrahedral cells.
```

```
      [vBase,nLDof,xLDof,sfun] = sf_tet_P3(i_eval,n_sdim,n_vert,varargin{:});
  end
else
  switch n_sdim
    case 2   % 2D Quadrilateral cells.
      [vBase,nLDof,xLDof,sfun] = sf_quad_Q3(i_eval,n_sdim,n_vert,varargin{:});
    case 3   % 3D Hedahedral cells.
      [vBase,nLDof,xLDof,sfun] = sf_hex_Q3(i_eval,n_sdim,n_vert,varargin{:});
  end
end
```

11.74 ellib/sflag4

SFLAG4 Shape function driver (fourth order P4/Q4 Lagrange polynomials).

[VBASE, NLDOF, XLDOF, SFUN] =

SFLAG4(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming quartic Lagrange shape functions with values defined in the nodes, and edges, (and also faces, and cell centers for quadrilaterals, tetrahedra, and hexahedrals).

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
```

```
sfun        string                  Function name of called shape function
```

See also

sf_line_P4, sf_tri_P4, sf_tet_P4, sf_quad_Q4, sf_hex_Q4

Code listing

```
if ( (n_vert==n_sdim+1) || (n_sdim==1) )   % Simplex cells (1D line, 2D triangle, 3D tetrahedron)
  switch n_sdim
    case 1   % 1D Line cells.
      [vBase,nLDof,xLDof,sfun] = sf_line_P4(i_eval,n_sdim,n_vert,varargin{:});
    case 2   % 2D Triangular cells.
      [vBase,nLDof,xLDof,sfun] = sf_tri_P4(i_eval,n_sdim,n_vert,varargin{:});
    case 3   % 3D Tetrahedral cells.
      [vBase,nLDof,xLDof,sfun] = sf_tet_P4(i_eval,n_sdim,n_vert,varargin{:});
  end
else
  switch n_sdim
    case 2   % 2D Quadrilateral cells.
      [vBase,nLDof,xLDof,sfun] = sf_quad_Q4(i_eval,n_sdim,n_vert,varargin{:});
    case 3   % 3D Hedahedral cells.
      [vBase,nLDof,xLDof,sfun] = sf_hex_Q4(i_eval,n_sdim,n_vert,varargin{:});
  end
end
```

11.75 ellib/sflag5

SFLAG5 Shape function driver (fifth order P5/Q5 Lagrange polynomials).

[VBASE, NLDOF, XLDOF, SFUN] =

SFLAG5(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates conforming quintic Lagrange shape functions with values defined in the nodes, and edges, (and also faces, and cell centers for quadrilaterals, tetrahedra, and hexahedrals).

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 1-3            Number of space dimensions
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_line_P5, sf_tri_P5, sf_tet_P5, sf_quad_Q5, sf_hex_Q5

Code listing

```
if ( (n_vert==n_sdim+1) || (n_sdim==1) )   % Simplex cells (1D line, 2D triangle, 3D tetrahedron)

  switch n_sdim

    case 1   % 1D Line cells.

      [vBase,nLDof,xLDof,sfun] = sf_line_P5(i_eval,n_sdim,n_vert,varargin{:});
```

```
      case 2   % 2D Triangular cells.
        [vBase,nLDof,xLDof,sfun] = sf_tri_P5(i_eval,n_sdim,n_vert,varargin{:});
      case 3   % 3D Tetrahedral cells.
        [vBase,nLDof,xLDof,sfun] = sf_tet_P5(i_eval,n_sdim,n_vert,varargin{:});
    end
  else
    switch n_sdim
      case 2   % 2D Quadrilateral cells.
        [vBase,nLDof,xLDof,sfun] = sf_quad_Q5(i_eval,n_sdim,n_vert,varargin{:});
      case 3   % 3D Hedahedral cells.
        [vBase,nLDof,xLDof,sfun] = sf_hex_Q5(i_eval,n_sdim,n_vert,varargin{:});
    end
  end
```

11.76 ellib/sfnoncon1

SFNONCON1 Shape function driver (first order P1/Q1 Lagrange polynomials).

[VBASE, NLDOF, XLDOF, SFUN] =

SFNONCON1(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates nonconforming linear (P-1/Q1∼) shape functions with degrees of freedom defined on the cell edges/faces.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 2-3            Number of space dimensions
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
```

```
nLDof       [4]                   Number of local degrees of freedom on
                                  vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]       Local coordinates of local dofs
sfun        string                Function name of called shape function
```

See also

sf_simp_P1nc, sf_quad_Q1nc, sf_hex_Q1nc

Code listing

```
if ( n_sdim==1 )

  error('sfnoncon1: 1D nonconforming shape functions not supported.')

elseif ( n_vert==n_sdim+1 )   % Simplex cells (2D triangle or 3D tetrahedron).

  [vBase,nLDof,xLDof,sfun] = sf_simp_P1nc(i_eval,n_sdim,n_vert,varargin{:});

else

  switch n_sdim

    case 2   % 2D Quadrilateral cells.

      [vBase,nLDof,xLDof,sfun] = sf_quad_Q1nc(i_eval,n_sdim,n_vert,varargin{:});

    case 3   % 3D Hedahedral cells.

      [vBase,nLDof,xLDof,sfun] = sf_hex_Q1nc(i_eval,n_sdim,n_vert,varargin{:});

  end
end
```

11.77 ellib/sfnonconp1

SFNONCONP1 Shape function driver (first order P1 Lagrange polynomials).

[VBASE, NLDOF, XLDOF, SFUN] =

SFNONCONP1(I_EVAL, N_SDIM, N_VERT, I_DOF, XI, AINVJAC, VBASE) Evaluates nonconforming linear (P-1/P1) shape functions with degrees of freedom defined on the cell edges/faces for simplices and interior for quadrilaterals/hexahedrals.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
i_eval      scalar:  1             Evaluate function values
                    >1             Evaluate values of derivatives
n_sdim      scalar: 2-3            Number of space dimensions
n_vert      scalar: 2-8            Number of vertices per cell
i_dof       scalar: 1-n_ldof       Local basis function to evaluate
xi          [n_sdim(+1)]           Local coordinates of evaluation point
aInvJac     [n,n_sdim(+1)*n_sdim]  Inverse of transformation Jacobian
vBase       [n]                    Preallocated output vector
                                                                            .
Output      Value/[Size]           Description
-----------------------------------------------------------------------------
vBase       [n]                    Evaluated function values
nLDof       [4]                    Number of local degrees of freedom on
                                   vertices, edges, faces, and cell interiors
xLDof       [n_sdim,n_ldof]        Local coordinates of local dofs
sfun        string                 Function name of called shape function
```

See also

sf_simp_P1nc, sf_disc1

Code listing

```
if ( n_sdim==1 )

  error('sfnonconp1: 1D nonconforming shape functions not supported.')

elseif ( n_vert==n_sdim+1 )   % Simplex cells (2D triangle or 3D tetrahedron).

  [vBase,nLDof,xLDof,sfun] = sf_simp_P1nc(i_eval,n_sdim,n_vert,varargin{:});

else

  [vBase,nLDof,xLDof,sfun] = sf_disc1(i_eval,n_sdim,n_vert,varargin{:});

end
```

11.78 ellib/lcoord_cell

LCOORD_CELL Local cell coordinates for the degrees of freedom.

[XI] =

LCOORD_CELL(NLDOF, N_SDIM, N_VC) Computes the local cell coordinates for all local degree of freedom. NLDOF is an array specifying the number of local degrees of freedom on vertices, edges, faces, and cell interiors. N_SDIM is the number of space dimensions, and N_VC the number of vertices per cell.

Code listing

```
% Local cell vertex coordinates.
 if( n_vc==(n_sdim+1) )
   xiv = eye(n_sdim+1);
 else
   if( n_sdim==2 )
     xiv = [ -1  1 1 -1 ;
             -1 -1 1  1 ];
   elseif( n_sdim==3 )
     xiv = [ -1  1  1 -1 -1  1 1 -1 ;
             -1 -1  1  1 -1 -1 1  1 ;
             -1 -1 -1 -1  1  1 1  1 ];
   end
 end


 xi = [];

 for j=1:size(nLDof,2)
   dim = j-1;

   for i=1:size(nLDof,1)
     n_ldof_ij = nLDof(i,j);
     if( n_ldof_ij==0 )
       continue   % Skip.
     end

     switch( dim )

       case 0   % Vertices.

         xi = [ xi xiv ];

       case 1   % Edges.
```

```
    n_ec = n_vc*( n_sdim==2 ) + (6*(n_vc==4) + 12*(n_vc==8))*( n_sdim==3 );

    xie = [];
    for k_ldof=1:n_ldof_ij
      k_edge = mod(k_ldof-1,n_ec) + 1;

      dt  = ceil(k_ldof/n_ec) * 1/((n_ldof_ij/n_ec)+1);
      xie = [ xie lcoorde( k_edge, dt, n_vc, n_sdim ) ];
    end

    xi = [ xi xie ];

  case 2   % Faces.

    n_f_c = 4*(n_vc==4) + 6*(n_vc==8);

    if( n_vc==4 )

      switch( n_ldof_ij )
        case 4
          dt = ones(3,1)/3;
        case 12
          dt = [ 1/2 1/4 1/4 ;
                 1/4 1/2 1/4 ;
                 1/4 1/4 1/2 ];
        case 24
          dt = [ 3/5 1/5 1/5 ;
                 2/5 2/5 1/5 ;
                 1/5 3/5 1/5 ;
                 1/5 2/5 2/5 ;
                 1/5 1/5 3/5 ;
                 2/5 1/5 2/5 ]';
        otherwise
          warning(['lcoord_cell: dof positions for ',int2str(n_ldof_ij),' face dofs not suppo
      end

    elseif( n_vc==8 )

      if( (n_ldof_ij/6)==4 )

        dt = [ 1 2 2 1; 1 1 2 2 ]/3;

      elseif( (n_ldof_ij/6)==9 )

        dt = [ 1 2 3 3 3 2 1 1 2; 1 1 1 2 3 3 3 2 2]/4;

      elseif( (n_ldof_ij/6)==16 )

        dt = [ 1 2 3 4 4 4 4 3 2 1 1 1 2 3 3 2; 1 1 1 1 2 3 4 4 4 4 3 2 2 2 3 3]/5;

      elseif( rem(sqrt(n_ldof_ij/6),1)==0 )
```

```
            xi1 = linspace( 0, 1, sqrt(n_ldof_ij/6) + 2 );
            [xi2,xi1] = meshgrid( xi1(2:end-1), xi1(2:end-1) );
            dt = [ xi1(:)'; xi2(:)' ];

          else
            warning(['lcoord_cell: dof positions for ',int2str(n_ldof_ij),' face dofs not support
          end

        end

        xif = [];
        for i_dt=1:size(dt,2)
          for k_f=1:n_f_c
            xif = [ xif lcoorde( k_f, dt(:,i_dt), n_vc, n_sdim ) ];
          end
        end

        xi = [ xi xif ];

      case 3   % Interior.

        xii = [];
        switch( n_sdim )

          case 1

            xi1 = linspace( 1, 0, n_ldof_ij+2 );
            xii = [ xi1(2:end-1); 1-xi1(2:end-1) ];

          case 2

            if( n_vc==3 )

              switch( n_ldof_ij )
                case 1
                  xii = ones(3,1)/3;
                case 3
                  xii = [ 1/2 1/4 1/4 ;
                          1/4 1/2 1/4 ;
                          1/4 1/4 1/2 ];
                case 6
                  xii = [ 3/5 1/5 1/5 2/5 1/5 2/5 ;
                          1/5 3/5 1/5 2/5 2/5 1/5 ;
                          1/5 1/5 3/5 1/5 2/5 2/5 ];
                otherwise
                  warning(['lcoord_cell: dof positions for ',int2str(n_ldof_ij),' interior face d
              end

            elseif( n_vc==4 )

              if( n_ldof_ij==3 )
                xii = zeros(2,3);
```

```
                elseif( n_ldof_ij==4 )   % Use counterclockwise ordering for 4 internal quad dofs.
                  xii = [ -1 1 1 -1; -1 -1 1 1 ]/3;
                elseif( n_ldof_ij==9 )
                  xii = [-1  1 1 -1  0 1 0 -1 0;-1 -1 1  1 -1 0 1  0 0]/2;
                elseif( n_ldof_ij==16 )
                  xii = [-3/5  3/5 3/5 -3/5  -1/5  3/5 1/5 -3/5   1/5 3/5 -1/5 -3/5
-1/5  1/5 1/5 -1/5;
                         -3/5 -3/5 3/5  3/5  -3/5 -1/5 3/5  1/5  -3/5 1/5  3/5 -1/5
-1/5 -1/5 1/5  1/5];
                elseif( rem(sqrt(n_ldof_ij),1)==0 )
                  xi1 = linspace( -1, 1, sqrt(n_ldof_ij)+2 );
                  [xi2,xi1] = meshgrid( xi1(2:end-1), xi1(2:end-1) );
                  xii = [ xi1(:)'; xi2(:)' ];
                else
                  error(['lcoord_cell: quadrilateral cells with ',num2str(n_ldof_ij),' internal dof
                end

              end

            case 3

              if( n_vc==4 )

                switch( n_ldof_ij )
                  case 1
                    xii = ones(4,1)/4;
                  case 4
                    xii = [ 2/5 1/5 1/5 1/5 ;
                            1/5 2/5 1/5 1/5 ;
                            1/5 1/5 2/5 1/5 ;
                            1/5 1/5 1/5 2/5 ];
                  otherwise
                    warning(['lcoord_cell: dof positions for ',int2str(n_ldof_ij),' interior face d
                end

              elseif( n_vc==8 )

                if( n_ldof_ij==4 )
                  xii = zeros(3,4);
                elseif( n_ldof_ij==8 )   % Use counterclockwise ordering for 8 internal hex dofs.
                  xii = [ -1 1 1 -1 -1 1 1 -1; -1 -1 1 1 -1 -1 1 1; -1 -1 -1 -1 1 1 1 1 ]/3;
                elseif( n_ldof_ij==64 )
                  xii = [-3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5
1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5
3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5 1/5 3/5 -3/5 -1/5 1/5 3/5 -3/5 -1/5
1/5  3/5 -3/5 -1/5  1/5  3/5 -3/5 -1/5 1/5 3/5 -3/5 -1/5 1/5 3/5;
                         -3/5 -3/5 -3/5 -3/5 -1/5 -1/5 -1/5 -1/5  1/5  1/5  1/5  1/5
3/5  3/5  3/5  3/5 -3/5 -3/5 -3/5 -3/5 -1/5 -1/5 -1/5 -1/5  1/5  1/5  1/5  1/5  3/5
3/5  3/5  3/5 -3/5 -3/5 -3/5 -3/5 -1/5 -1/5 -1/5 -1/5  1/5  1/5 1/5 1/5  3/5  3/5 3/5 3/5 -3/5 -3/
1/5  1/5 1/5 1/5  3/5  3/5 3/5 3/5;
                         -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5 -3/5
1/5  1/5  1/5  1/5  1/5  1/5  1/5  1/5  1/5  1/5 1/5 1/5  1/5  1/5 1/5 1/5  3/5  3/5
```

```
3/5  3/5  3/5  3/5  3/5  3/5  3/5  3/5 3/5 3/5  3/5  3/5 3/5 3/5];
                elseif( rem(n_ldof_ij^(1/3),1)==0 )
                  xi1 = linspace( -1, 1, n_ldof_ij^(1/3)+2 );
                  [xi2,xi1,xi3] = meshgrid( xi1(2:end-1), xi1(2:end-1), xi1(2:end-1) );
                  xii = [ xi1(:)'; xi2(:)'; xi3(:)' ];
                else
                  error(['lcoord_cell: hexahedral cells with ',num2str(n_ldof_ij),' internal dofs n
                end

              end

          end

          xi = [ xi xii ];

      end

    end
  end
```

11.79 ellib/grid_cell_type

GRID_CELL_TYPE Determine grid cell type.

[CELL_TYPE, N_SDIM, N_VC] =

GRID_CELL_TYPE(VARARGIN) Determines the cell type from cell vertex connectivity array C, or alternatively the number of space dimensions N_SDIM, and the number of vertices per cell N_VC. Returns the CELL_TYPE, N_SDIM, and N_VC. Also assigns the cell_type integer code variables (LINE, TRI, QUAD, TET, and HEX) in the workspace of the caller function.

Code listing

```
assignin( 'caller', 'LINE', 12 );
assignin( 'caller', 'TRI',  23 );
assignin( 'caller', 'QUAD', 24 );
assignin( 'caller', 'TET',  34 );
assignin( 'caller', 'HEX',  38 );


if( nargin>=1 && nargin<=2 && all(cellfun(@isnumeric,varargin)) )

  if( nargin==1 && numel(varargin{1})>1 )
```

```
      c = varargin{1};
      n_vc = size(c,1);
      if( n_vc==4 )
        if( size(c,2)==1 )
          warning(['Cannot separate from quadrilateral or tetrahedral cell.',char(10),'Defaulting t
        end

        [~,tmp] = unique( sort( [ c(1:3,:) c([1 2 4],:) c([2 3 4],:) c([3 1 4],:) ], 1 )', 'rows' )
        is_quad = numel(tmp) == 4*size(c,2);
        n_sdim = 2 + ~is_quad;
      else
        n_sdim = (n_vc==2) + 2*(n_vc==3) + 3*(n_vc==8);
      end

      varargin = { n_sdim n_vc };

    end

    cell_type = cell2mat( varargin ) * [10.^[(numel(varargin)-1):-1:0]]';

    n_sdim = floor(cell_type/10);
    n_vc   = rem(cell_type,10);

  end
```

11.80 evalexpr

EVALEXPR Evaluates an expression in points.

[VEVAL] =

EVALEXPR(S_EXPR, XP, PROB, SOLNUM, VEC_EVAL) Evaluates the expression S_EXPR in the points specified in XP (in global x/y/z-coordinates). PROB is a valid finite element problem struct. The VEC_EVAL flag (default true) toggles vectorized evaluation for dof variables with P0-P2 fem shape functions.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
s_expr      string                 String expression to evaluate
xp          [n_sdim,n_xp]          Coordinates of evaluation points
prob        struct                 Problem definition struct
solnum      scalar {n_sols}        Solution number/time to evaluate
vec_eval    logical {true}         Use vectorized evaluation for dep. variables
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
```

```
vEval       [n_xp,1]              Output vector of evaluated values
```

11.81 examples/ex_brinkman1

EX_BRINKMAN1 Coupled Navier-Stokes and Brinkman equations model.

[FEA, OUT] =

EX_BRINKMAN1(VARARGIN) Example using the Navier-Stokes equations coupled with the Brinkman equations for porous media flow.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {2e-4}          Grid size
sfun        string {sflag2}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'hmax',     4e-4;
             'iplot',    1;
             'tol',      0.25;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Model constants.
 rho = 1.23;     % Density.
 eta = 2.1e-4;   % Viscosity.
 kap = 3.2e-7;   % Permeability.


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' };
 fea.geom.objects = { gobj_rectangle(-2e-3,0,-8e-3,8e-3,'R1') ...
```

```
                      gobj_polygon([0 -6e-3;2e-3 -5e-3;2e-3 4e-3;0 6e-3],'P1') ...
                                  };
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', opt.fid );
 fea.grid = gridbdrx( fea.grid );   %  Reconstruct internal boundaries.


% Problem definition.

% Navier-Stokes equations.
 fea = addphys( fea, @navierstokes );   % Add Navier-Stokes equations physics mode.

 fea.phys.ns.eqn.coef{1,end} = rho;
 fea.phys.ns.eqn.coef{2,end} = eta;

 fea.phys.ns.bdr.sel(1) = 2;
 fea.phys.ns.bdr.coef{2,end}{2,1} = 2e-2;   % Set inflow velocity.
 fea.phys.ns.bdr.sel(4) = 4;
 fea.phys.ns.bdr.sel(9) = 2;
 fea.phys.ns.bdr.coef{2,end}{1,9} = 'u2';   % Brinkman velocity from shared boundary.
 fea.phys.ns.bdr.coef{2,end}{2,9} = 'v2';

% Brinkman equations.
 fea = addphys( fea, @brinkmaneqns );   % Add Brinkman equations physics mode.

 fea.phys.br.eqn.coef{1,end} = rho;
 fea.phys.br.eqn.coef{2,end} = eta;
 fea.phys.br.eqn.coef{3,end} = kap;

 fea.phys.br.bdr.sel(9) = 2;
 fea.phys.br.bdr.coef{2,end}{1,9} = 'u';   % NS velocity from shared boundary.
 fea.phys.br.bdr.coef{2,end}{2,9} = 'v';


% Parse and solve problem.
 fea = parsephys( fea );
 fea = parseprob( fea );
 fea.sol.u = solvestat( fea, 'solcomp', {{1 0} {1 0} {1 0} {0 1} {0 1} {0 1} }, 'fid', opt.fid );


% Postprocessing.
 x = linspace( -1e-3, 1e-3, 100 );
 y = zeros(size(x));
 u_ns = evalexpr( 'sqrt(u^2+v^2)', [x;y], fea );
 u_br = evalexpr( 'sqrt(u2^2+v2^2)', [x;y], fea );
 if( opt.iplot )
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'sqrt(u^2+v^2)', 'arrowexpr', {'u' 'v'} )
   postplot( fea, 'surfexpr', 'sqrt(u2^2+v2^2)', 'arrowexpr', {'u2' 'v2'} )

   subplot(1,2,2)
   plot( x, u_ns ), hold on
   plot( x, u_br ), hold on
```

```
 end


% Error checking.
 out.err = [ abs(max( u_ns ) - 0.0175)/0.0175, ...
             abs(mean( u_br(~isnan(u_br)) ) - 9e-3)/9e-3];
 out.pass = all( out.err < opt.tol );


 if ( nargout==0 )
   clear fea out
 end
```

11.82 examples/ex_convdiff1

EX_CONVDIFF1 2D Convection and diffusion equation example on a rectangle.

[FEA, OUT] =

EX_CONVDIFF1(VARARGIN) Convection and diffusion equation on a rectangle with exact solution u_0+c1∗eta+c2∗(2∗cd∗xi+eta^2). Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {1/40}          Max grid cell size
a           scalar {1}             Convection velocity in x-direction
b           scalar {2}             Convection velocity in y-direction
c1          scalar {1}             Solution constant
c2          scalar {0.8}           Solution constant
cd          scalar {0.5}           Diffusion coefficient
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'igrid',    0; ...
   'hmax',     1/40; ...
   'a',        1; ...
   'b',        2; ...
   'c1',       1; ...
   'c2',       0.8; ...
   'cd',       0.5; ...
   'sfun',     'sflag1'; ...
   'iphys',    1; ...
   'iplot',    1; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;
 xi        = [num2str(opt.a),'*x+',num2str(opt.b),'*y'];
 eta       = [num2str(opt.b),'*x-',num2str(opt.a),'*y'];
 refsol    = [num2str(opt.c1),'*(',eta,')+',num2str(opt.c2),'*(2*',num2str(opt.cd),'*(',xi,')+(',e


% Geometry definition.
 gobj = gobj_rectangle();
 fea.geom.objects = { gobj };


% Grid generation.
 switch opt.igrid
   case -1
     fea.grid = rectgrid(round(1/opt.hmax));
     fea.grid = quad2tri(fea.grid);
   case 0
     fea.grid = rectgrid(round(1/opt.hmax));
   case 1
     fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 end
 n_bdr = max(fea.grid.b(3,:));           % Number of boundaries.


% Problem definition.
 fea.sdim  = { 'x' 'y' };                % Coordinate names.
 if ( opt.iphys==1 )

   fea = addphys(fea,@convectiondiffusion);   % Add convection and diffusion physics mode.
   fea.phys.cd.sfun = { opt.sfun };           % Set shape function.
   fea.phys.cd.eqn.coef{2,4}   = { opt.cd };  % Set diffusion coefficient.
   fea.phys.cd.eqn.coef{3,4}   = { opt.a  };  % Convection velocity in x-direction.
   fea.phys.cd.eqn.coef{4,4}   = { opt.b  };  % Convection velocity in y-direction.
   fea.phys.cd.bdr.sel         = [1 1 1 1];
   fea.phys.cd.bdr.coef{1,end} = repmat({refsol},1,n_bdr);   % Set Dirichlet boundary coefficient
   fea = parsephys(fea);                  % Check and parse physics modes.

 else
```

```
    fea.dvar  = { 'c' };                  % Dependent variable name.
    fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
    fea.eqn.a.form = { [2 3 2 3;2 3 1 1] };   % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
    fea.eqn.a.coef = { [opt.cd opt.cd opt.a opt.b] };   % Coefficients used in assembling stiffness

    fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
    fea.eqn.f.coef = { 0 };               % Coefficient used in right hand side.

% Define boundary conditions.
    fea.bdr.d     = cell(1,n_bdr);
   [fea.bdr.d{:}] = deal(refsol);         % Assign reference solution to all boundaries (Dirichlet)

    fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid);   % Call to stationary solver.


% Postprocessing.
 s_err = ['abs(',refsol,'-c)'];
 if ( opt.iplot>0 )
   figure
   subplot(2,1,1)
   postplot(fea,'surfexpr','c','isoexpr','c')
   title('Solution c')
   subplot(2,1,2)
   postplot(fea,'surfexpr',s_err)
   title('Error')
 end


% Error checking.
 if ( size(fea.grid.c,1)==4 )
   xi = [0;0];
 else
   xi = [1/3;1/3;1/3];
 end
 err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
 ref = evalexpr0('c',xi,1,1:size(fea.grid.c,2),[],fea);
 err = sqrt(sum(err.^2)/sum(ref.^2));


 if( ~isempty(fid) )
```

```
  fprintf(fid,'\nL2 Error: %f\n',err)
  fprintf(fid,'\n\n')
end


out.err  = err;
out.pass = out.err<0.1;
if ( nargout==0 )
  clear fea out
end
```

11.83 examples/ex_convdiff2

EX_CONVDIFF2 1D Time dependent convection and diffusion equation example.

[FEA, OUT] =

EX_CONVDIFF2(VARARGIN) 1D time dependent convection and diffusion equation on a line with exact solution exp(-k^2*nu*t)*sin(k*(x-a*t)) and periodic boundary conditions. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
k           scalar {2*pi}          Simulation parameter
a           scalar {1}             Convection velocity
nu          scalar {0.1}           Diffusion coefficient
hmax        scalar {1/25}          Max grid cell size
dt          scalar {0.01}          Time step size
ischeme     scalar {2}             Time stepping scheme
sfun        string {sflag1}        Shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
cOptDef = { ...
  'k',        2*pi; ...
  'a',        1; ...
```

```
    'nu',       0.1; ...
    'hmax',     1/25; ...
    'dt'        0.01; ...
    'ischeme'   2; ...
    'sfun',     'sflag1'; ...
    'iplot',    1; ...
    'tol',      1e-1; ...
    'fid',      1 };
  [got,opt] = parseopt(cOptDef,varargin{:});
  fid       = opt.fid;


  refsol = ['exp(-',num2str(opt.k^2*opt.nu),'*t)*sin(',num2str(opt.k),'*(x-',num2str(opt.a),'*t))']


% Grid generation.
  fea.grid = linegrid( 1/opt.hmax, 0, 1 );


% Problem definition.
  fea.sdim  = { 'x' };
  fea = addphys( fea, @convectiondiffusion );
  fea.phys.cd.sfun = { opt.sfun };
  fea.phys.cd.eqn.coef{2,4} = { opt.nu };
  fea.phys.cd.eqn.coef{3,4} = { opt.a  };
  fea = parsephys(fea);


% Parse and solve problem.
  fea = parseprob( fea );


  x = fea.grid.p';
  n = length(x);
  if( strcmp( opt.sfun,'sflag2' ) )
    x = [ x; (x(2:end)+x(1:end-1))/2 ];
  end
  t  = 0;
  u0 = eval( refsol );


% Assembly.
  [M,A,f] = assembleprob( fea, 'f_m', 1, 'imass', 1, 'f_a', 1, 'f_f', 1, 'f_sparse', 1 );
  M = spdiags( full(sum(M')'), 0, size(M,1), size(M,1) );
  fea.sol.u = u0;
  dt = opt.dt;
  it = 0;
  tlist = 0;
  tmax = 1;
  if( opt.ischeme==2 )       % Crank-Nicolson.
    C = [ M + dt/2*A ];
  elseif( opt.ischeme==1 )   % Backward Euler.
```

```
    C = [ M + dt*A ];
  end
  v0 = zeros(size(C,1),1);
  v0(1) =  1;
  v0(n) = -1;
  C = [C v0; v0' 0];


% Solver loop.
  while 1
    t  = t  + dt;
    it = it + 1;

    u_r = eval( refsol );
    if( opt.ischeme==2 )       % Crank-Nicolson.
      b = [ [ M - dt/2*A ]*u0 + dt*f ];
    elseif( opt.ischeme==1 )   % Backward Euler.
      b = [ M*u0 + dt*f ];
    end

    u1 = C\[b;0];
    u1(end) = [];

    if( t>=tmax )
      break
    end

    err = norm( u1 - u_r )/norm( u_r );
    errnm(it) = err;
    if( ~isempty(fid) )
      fprintf( fid, 'Time = %f, error norm = %d\n', t, err );
    end

    fea.sol.u = [ fea.sol.u u1 ];
    tlist = [ tlist t ];
    u0 = u1;
  end


% Postprocessing.
  if( opt.iplot>0 )
    figure;
    if( opt.iplot>1 )
      i_sol_list = 1:numel(tlist);
    else
      i_sol_list = numel(tlist);
    end
    [~,ix] = sort( x );
    for i_sol=i_sol_list
      t = tlist(i_sol);
      clf
      postplot( fea, 'surfexpr', 'c', 'solnum', i_sol );
```

```
      hold on
      u_r = eval( refsol );
      plot( sort(x), u_r(ix), 'r--' );
      title( ['Solution at time ',num2str(t)])
      xlabel( 'x' )
      drawnow
    end
  end


% Error checking.
  out.err  = errnm;
  out.pass = all( errnm<opt.tol );


  if ( nargout==0 )
    clear fea out
  end
```

11.84 examples/ex_convdiff3

EX_CONVDIFF3 1D Time dependent convection and diffusion equation example.

[FEA, OUT] =

EX_CONVDIFF3(VARARGIN) 1D time dependendt convection and diffusion equation on a line with an infinately oscillating exact solution. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
w           scalar {1.5*pi}        Simulation parameter
U           scalar {2}             Simulation parameter
a           scalar {1}             Convection velocity
nu          scalar {0.1}           Diffusion coefficient
hmax        scalar {1/25}          Max grid cell size
dt          scalar {0.02}          Time step size
ischeme     scalar {3}             Time stepping scheme
sfun        string {sflag2}        Shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'w',        1.5*pi; ...
   'U',        2; ...
   'a',        1; ...
   'nu',       0.1; ...
   'hmax',     1/25; ...
   'dt'        0.02; ...
   'ischeme'   3; ...
   'sfun',     'sflag1'; ...
   'iplot',    1; ...
   'tol',      3e-2; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


 w  = opt.w;
 U  = opt.U;
 a  = opt.a;
 nu = opt.nu;
 l1 = ['(',num2str(a),'+sqrt(',num2str(a^2),'+',num2str(4*nu*w),'*i))/',num2str(2*nu)];
 l2 = ['(',num2str(a),'-sqrt(',num2str(a^2),'+',num2str(4*nu*w),'*i))/',num2str(2*nu)];
 refsol  = ['(exp(',l1,'*x)-exp(',l2,'*x))/(exp(',l1,')-exp(',l2,'))*',num2str(U),'*exp(i*',num2st
 refsol0 = ['(exp(',l1,'*x)-exp(',l2,'*x))/(exp(',l1,')-exp(',l2,'))*',num2str(U)];


% Grid generation.
 fea.grid = linegrid( 1/opt.hmax, 0, 1 );


% Problem definition.
 fea.sdim  = { 'x' };
 fea = addphys( fea, @convectiondiffusion );
 fea.phys.cd.sfun = { opt.sfun };
 fea.phys.cd.eqn.coef{2,4} = { opt.nu };
 fea.phys.cd.eqn.coef{3,4} = { opt.a  };
 fea = parsephys(fea);


% Parse and solve problem.
 fea = parseprob( fea );
 fea.bdr.d{1} = 0;
 fea.bdr.d{2} = [num2str(U),'*cos(',num2str(w),'*t)'];
 fea.bdr.n    = cell(1,2);
 x = fea.grid.p';
 if( strcmp( opt.sfun,'sflag2' ) )
   x = [ x; (x(2:end)+x(1:end-1))/2 ];
 end
```

```
  init = real( eval( refsol0 ) );
  [fea.sol.u,tlist] = solvetime( fea, 'fid', fid, 'init', init, 'ischeme', opt.ischeme, 'tstep', op


% Postprocessing.
 if( opt.iplot>0 )
   figure;
   if( opt.iplot>1 )
     i_sol_list = 1:numel(tlist);
   else
     i_sol_list = numel(tlist);
   end
   [~,ix] = sort( x );
   for i_sol=i_sol_list
     t = tlist(i_sol);
     clf
     postplot( fea, 'surfexpr', 'c', 'solnum', i_sol );
     hold on
     u_r = real( eval( refsol ) );
     plot( sort(x), u_r(ix), 'r--' );
     title( ['Solution at time ',num2str(t)])
     xlabel( 'x' )
     drawnow
   end
 end


% Error checking.
 for i_sol=1:numel(tlist)
   u_i = fea.sol.u(:,i_sol);
   t   = tlist(i_sol);
   u_r = real( eval( refsol ) );
   errnm(i_sol) = norm( u_i - u_r )/norm( u_r );
 end
 out.err  = errnm;
 out.pass = all( errnm<opt.tol );


 if ( nargout==0 )
   clear fea out
 end
```

11.85 examples/ex_convdiff4

EX_CONVDIFF4 1D Burgers equation (convection and diffusion) example.

[FEA, OUT] =

EX_CONVDIFF4(VARARGIN) 1D Burgers equation with steady solution, u_t + (b∗u-c)∗u_x - nu∗u_xx = 0 with exact solution c/b∗(1-tanh(c/(2∗nu)∗(x-x0))). Tests both time dependent and steady solvers. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
b           scalar {1.0}           Strength of nonlinearity
c           scalar {0.13}          Convection velocity
nu          scalar {0.01}          Diffusion coefficient
x0          scalar {0.5}           Posision of smooth shock
hmax        scalar {1/25}          Max grid cell size
ischeme     scalar {-1}            Solver scheme (<0 = stationary)
nsolve      scalar {2}             Nonlinear solver (when ischeme<0)
dt          scalar {0.1}           Time step size
sfun        string {sflag1}        Shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
             'b',        1.0; ...
             'c',        0.13; ...
             'nu',       0.01; ...
             'x0',       0.5; ...
             'hmax',     1/20; ...
             'ischeme'   -1; ...
             'nsolve'    2; ...
             'dt'        0.1; ...
             'sfun',     'sflag1'; ...
             'iplot',    1; ...
             'tol',      1e-2; ...
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


 b  = opt.b;
 c  = opt.c;
 nu = opt.nu;
 x0 = opt.x0;
 refsol = [num2str(c/b),'*(1-tanh(',num2str(c/(2*nu)),'*(x-',num2str(x0),')))'];
```

```
 bcd1   = c/b*(1-tanh((c/(2*nu)*(0-x0))));
 bcd2   = c/b*(1-tanh((c/(2*nu)*(1-x0))));

% Grid generation.
 fea.grid = linegrid( 1/opt.hmax, 0, 1 );


% Problem definition.
 fea.sdim  = { 'x' };
 fea = addphys( fea, @convectiondiffusion );
 fea.phys.cd.sfun = { opt.sfun };
 fea.phys.cd.eqn.coef{2,4} = { opt.nu };
 fea.phys.cd.eqn.coef{3,4} = { [num2str(b),'*c-',num2str(c)] };
 fea = parsephys(fea);


% Parse and solve problem.
 fea = parseprob( fea );
 fea.bdr.d{1} = bcd1;
 fea.bdr.d{2} = bcd2;
 fea.bdr.n    = cell(1,2);
 x = fea.grid.p';
 if( strcmp( opt.sfun,'sflag2' ) )
   x = [ x; (x(2:end)+x(1:end-1))/2 ];
 end
 if( opt.ischeme<0 )
   init = 0;
   jac.form  = {[1;1]};
   jac.coef  = {[num2str(b),'*cx']};
   fea.sol.u = solvestat( fea, 'fid', fid, 'init', init, 'maxnit', 1000, 'nsolve', 2, 'jac', jac,
 else
   init = [num2str(bcd1),'+x*',num2str(bcd2-bcd1)];
   [fea.sol.u,tlist] = solvetime( fea, 'fid', fid, 'init', init, 'ischeme', opt.ischeme, 'tstep',
 end


% Postprocessing.
 if( opt.iplot>0 )
   figure
   i_sol = size(fea.sol.u,2);
   [~,ix] = sort( x );
   postplot( fea, 'surfexpr', 'c', 'solnum', i_sol );
   hold on
   u_r = real( eval( refsol ) );
   plot( sort(x), u_r(ix), 'r--' );
   title( 'Steady solution' )
   xlabel( 'x' )
   drawnow
 end


% Error checking.
```

```
  i_sol = size(fea.sol.u,2);
  u_i   = fea.sol.u(:,i_sol);
  u_r   = real( eval( refsol ) );
  errnm = norm( u_i - u_r )/norm( u_r );
  out.err  = errnm;
  out.pass = all( errnm<opt.tol );


  if ( nargout==0 )
    clear fea out
  end
```

11.86 examples/ex_convdiff5

EX_CONVDIFF5 2D Convection and diffusion equation with high Peclet number.

[FEA, OUT] =

EX_CONVDIFF5(VARARGIN) Convection and diffusion equation on a unit square with high Peclet (Cell Reynolds) number requiring artificial stabilization/numerical diffusion. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
igrid       scalar {0}/1           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {1/20}          Max grid cell size
a           scalar {cos(pi/3)}     Convection velocity in x-direction
b           scalar {sin(pi/3)}     Convection velocity in y-direction
cd          scalar {1e-4}          Diffusion coefficient
sfun        string {sflag1}        Shape function
artstab     scalar {1}/0/2         Artificial stabilization (0=no stabilization)
                                   1=isotropic diffusion, 2=streamline diffusion
iplot       scalar {1}/0           Plot solution (=1)
                                                                               .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
  cOptDef = { ...
```

```
    'igrid',    0; ...
    'hmax',     1/20; ...
    'a',        cos(pi/3); ...
    'b',        sin(pi/3); ...
    'cd',       1e-4; ...
    'sfun',     'sflag1'; ...
    'artstab',  1; ...
    'iplot',    1; ...
    'tol',      0.1; ...
    'fid',      1 };
  [got,opt] = parseopt( cOptDef, varargin{:} );
  fid       = opt.fid;


% Geometry definition and grid generation.
  fea.geom.objects = { gobj_rectangle() };
  switch opt.igrid
    case -1
      fea.grid = rectgrid( round(1/opt.hmax) );
      fea.grid = quad2tri( fea.grid);
    case 0
      fea.grid = rectgrid( round(1/opt.hmax) );
    case 1
      fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );
  end


% Problem definition.
  fea.sdim = { 'x' 'y' };                      % Coordinate names.
  fea = addphys( fea, @convectiondiffusion ); % Add convection and diffusion physics mode.
  fea.phys.cd.sfun            = { opt.sfun }; % Set shape function.
  fea.phys.cd.eqn.coef{2,4}   = { opt.cd };   % Set diffusion coefficient.
  fea.phys.cd.eqn.coef{3,4}   = { opt.a  };   % Convection velocity in x-direction.
  fea.phys.cd.eqn.coef{4,4}   = { opt.b  };   % Convection velocity in y-direction.
  fea.phys.cd.eqn.coef{5,4}   = { 1 };
  fea.phys.cd.bdr.sel         = [1 1 1 1];
  fea.phys.cd.bdr.coef{1,end} = {1 0 0 1};


% Numerical stabilization.
  switch opt.artstab
    case 1
      fea.phys.cd.prop.artstab.id      = 1;
      fea.phys.cd.prop.artstab.id_coef = 0.5;
    case 2
      fea.phys.cd.prop.artstab.sd      = 1;
      fea.phys.cd.prop.artstab.sd_coef = 0.25;
  end


% Parse and solve problem.
  fea       = parsephys( fea );               % Check and parse physics modes.
```

```
  fea       = parseprob( fea );               % Check and parse problem struct.
  fea.sol.u = solvestat( fea, 'fid', fid );   % Call to stationary solver.


% Postprocessing.
 x = linspace( 0, 1, 2*(1/opt.hmax)+1 );
 y = 0.8*ones(size(x));
 c0p8 = evalexpr( 'c', [x;y], fea );
 c_ref = 1 + 0.1/0.05*x';
 c_ref(c_ref>1.9275) = 1.9275;
 if( opt.iplot>0 )
   figure
   subplot( 1, 2, 1 )
   postplot( fea, 'surfexpr', 'c', 'isoexpr', 'c' )
   title( 'Solution c' )
   subplot( 1, 2, 2 )
   plot( x, c_ref, 'r' )
   hold on
   plot( x, c0p8, 'b--' )
   title( 'Solution at y = 0.8' )
   xlabel( 'x' )
   ylabel( 'c' )
   grid on
 end


% Error checking.
 ix = find(x<=0.9);
 out.err  = norm( c_ref(ix) - c0p8(ix) )./norm( c_ref(ix) );
 out.pass = out.err < opt.tol;


 if( nargout==0 )
   clear fea out
 end
```

11.87 examples/ex_convdiff6

EX_CONVDIFF6 1D Stationary convection and diffusion equation example.

[FEA, OUT] =

EX_CONVDIFF6(VARARGIN) 1D stationary convection and diffusion equation on a line L with the exact solution u(x) = c0 + f/v∗x + f∗D/v^2∗(exp(-v∗L/D) - exp((x-L)∗v/D)). A Dirichlet condition c0 is prescribed at x=0 and at x=L is left free dc(x=L)/dx = 0. Accepts the following property/value pairs.

[1] Case B3 in M. Th. van Genuchten and W. J. Alves, Analytical Solutions to the One-Dimensional Convective-Dispersive Solution Transport Equation, pp. 29, Technical Bulletin 1661, Publisher: U.S. Department of Agriculture, 1982.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
D           scalar {2}             Diffusion coefficient
v           scalar {3}             Convection velocity
f           scalar {4}             Reaction source term
c0          scalar {5}             Dirichlet boundary condition at x=0
L           scalar {6}             Length of domain
nx          scalar {25}            Number of grid cells
sfun        string {sflag1}        Shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'D',     2; ...
             'v',     3; ...
             'f',     4; ...
             'c0',    5; ...
             'L',     6; ...
             'nx',    25; ...
             'sfun',  'sflag1'; ...
             'iplot', 1; ...
             'tol',   1e-2; ...
             'fid',   1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;


% Grid generation.
 fea.grid = linegrid( opt.nx, 0, opt.L );
```

```
% Problem definition.
 fea.sdim  = { 'x' };

 fea = addphys( fea, @convectiondiffusion );
 fea.phys.cd.sfun = { opt.sfun };

 fea.phys.cd.eqn.coef{2,4} = { opt.D };
 fea.phys.cd.eqn.coef{3,4} = { opt.v };
 fea.phys.cd.eqn.coef{4,4} = { opt.f };

 fea.phys.cd.bdr.sel(1)         = 1;
 fea.phys.cd.bdr.coef{1,end}{1} = opt.c0;


% Parse and solve problem.
 fea = parsephys( fea );
 fea = parseprob( fea );
 fea.sol.u = solvestat( fea, 'icub', 1+str2num(strrep(opt.sfun,'sflag','')), 'fid', opt.fid );



% Postprocessing.
 refsol = sprintf( '%d + %d*x + %d*( exp(-%d) - exp((x-%d)*%d) )', ...
                   opt.c0, opt.f/opt.v, opt.f*opt.D/opt.v^2, ...
                   opt.v*opt.L/opt.D, opt.L, opt.v/opt.D );

 x     = linspace( 0, opt.L,  100 );
 c     = evalexpr( 'c',    x, fea );
 c_ref = evalexpr( refsol, x, fea );
 if( opt.iplot>0 )
   plot( x, c,     'b-.', 'linewidth', 2 );
   hold on, grid on
   plot( x, c_ref, 'r--', 'linewidth', 1 );
   legend( 'computed solution', 'reference solution', 'location', 'southeast' )
 end


% Error checking.
 ix = find( c_ref~=0 );
 errnm    = norm( (c(ix) - c_ref(ix)) ./ c_ref(ix) );
 out.err  = errnm;
 out.pass = all( errnm<opt.tol );

 if( nargout==0 )
   clear fea out
 end
```

11.88 examples/ex_convreact1

EX_CONVREACT1 1D Time dependent convection-reaction equation example.

[FEA, OUT] =

EX_CONVREACT1(VARARGIN) 1D time dependent convection equation with reaction on a line with exact solutions. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
k           scalar {1e-1}          Reaction coefficient
a           scalar {1}             Convection velocity
hmax        scalar {1/25}          Max grid cell size
dt          scalar {0.1}           Time step size
ischeme     scalar {3}             Time stepping scheme
sfun        string {sflag1}        Shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'k',       -1; ...
   'a',        1; ...
   'hmax',     1/25; ...
   'dt'        0.1; ...
   'ischeme'   3; ...
   'sfun',     'sflag1'; ...
   'iplot',    1; ...
   'tol',      1e-2; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


 tmax   = 1;
 refsol = ['exp(',num2str(opt.k),'*t)*sin(x-',num2str(opt.a),'*t)'];


% Grid generation.
 fea.grid = linegrid( 1/opt.hmax, 0, 1 );
```

```
% Problem definition.
 fea.sdim  = { 'x' };
 fea = addphys( fea, @convectiondiffusion );
 fea.phys.cd.sfun = { opt.sfun };
 fea.phys.cd.eqn.coef{2,4} = { 0 };
 fea.phys.cd.eqn.coef{3,4} = { opt.a };
 fea.phys.cd.eqn.coef{4,4} = { [num2str(opt.k),'*c'] };
 fea = parsephys(fea);


% Parse and solve problem.
 x = fea.grid.p';
 n = length(x);
 if( strcmp( opt.sfun,'sflag2' ) )
   x = [ x; (x(2:end)+x(1:end-1))/2 ];
 end
 t   = 0;
 u0  = eval( refsol );
 fea = parseprob( fea );
 if( opt.ischeme>0 )

   fea.bdr.d{1} = refsol;
   fea.bdr.d{2} = refsol;
   fea.bdr.n    = cell(1,2);
   [fea.sol.u,tlist] = solvetime( fea, 'fid', fid, 'init', u0, 'ischeme', opt.ischeme, 'tstep', op

 else

   fea.sol.u = u0;
   [M,A] = assembleprob( fea, 'f_m', 1, 'imass', 2, 'f_a', 1, 'f_sparse', 1 );
   M = spdiags( full(sum(M')'), 0, size(M,1), size(M,1) );
   dt = opt.dt;
   it = 0;
   tlist = 0;
   if( opt.ischeme==-1 )       % Crank-Nicolson.
     C = [ M + dt/2*A ];
   elseif( opt.ischeme==-2 )   % Backward Euler.
     C = [ M + dt*A ];
   end
   C(1,:) = 0;
   C(1,1) = 1;
   C(n,:) = 0;
   C(n,n) = 1;
   fea_it = fea;
   while 1
     t  = t  + dt;
     it = it + 1;

     fea_it.sol.u = u0;
     [~,~,f0] = assembleprob( fea_it, 'f_f', 1 );
```

```
    u_r = eval( refsol );
    dnm = inf;
    while( dnm>1e-6 )
      [~,~,f1] = assembleprob( fea_it, 'f_f', 1 );

      if( opt.ischeme==-1 )       % Crank-Nicolson.
        b = [ [ M - dt/2*A ]*u0 + dt/2*f1 + dt/2*f0 ];
      elseif( opt.ischeme==-2 )   % Backward Euler.
        b = [ M*u0 + dt*f1 ];
      end
      b(1) = u_r(1);
      b(n) = u_r(n);

      dnm = fea_it.sol.u;
      u1  = C\b;
      dnm = norm( dnm - u1 );
      fea_it.sol.u = u1;

    end

    if( t>=tmax )
      break
    end

    err = norm( u1 - u_r )/norm( u_r );
    errnm(it) = err;
    if( ~isempty(fid) )
      fprintf( fid, 'Time = %f, error norm = %d\n', t, err );
    end

    fea.sol.u = [ fea.sol.u u1 ];
    tlist = [ tlist t ];
    u0 = u1;
  end

 end


% Postprocessing.
 if( opt.iplot>0 )
   figure;
   if( opt.iplot>1 )
     i_sol_list = 1:numel(tlist);
   else
     i_sol_list = numel(tlist);
   end
   [~,ix] = sort( x );
   for i_sol=i_sol_list
     t = tlist(i_sol);
     clf
     postplot( fea, 'surfexpr', 'c', 'solnum', i_sol );
```

```
      hold on
      u_r = eval( refsol );
      plot( sort(x), u_r(ix), 'r--' );
      title( ['Solution at time ',num2str(t)])
      xlabel( 'x' )
    end
  end


% Error checking.
  for i_sol=1:numel(tlist)
    u_i = fea.sol.u(:,i_sol);
    t   = tlist(i_sol);
    u_r = eval( refsol );
    errnm(i_sol) = norm( u_i - u_r )/norm( u_r );
  end
  out.err  = errnm;
  out.pass = all( errnm<opt.tol );


  if ( nargout==0 )
    clear fea out
  end
```

11.89 examples/ex_custom_equation1

EX_CUSTOM_EQUATION1 1D Black-Scholes custom equation example.

[FEA, OUT] =

EX_CUSTOM_EQUATION1(VARARGIN) 1D Black-Scholes model equation example using the custom equation physics mode. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
icase       scalar {1}/2           Test case equation to solve
tmax        scalar {1}             Maximum/stopping time
len         scalar {1}             Length of domain
hmax        scalar {1/20}          Grid cell size
ischeme     scalar {3}             Time stepping scheme
sfun        string {sflag1}        Finite element shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
  cOptDef = { ...
    'icase',    1; ...
    'tmax',     1; ...
    'len',      1; ...
    'hmax',     1/20; ...
    'ischeme'   3; ...
    'sfun',     'sflag1'; ...
    'iplot',    1; ...
    'dvname',   'u'; ...
    'tol',      1.1e-2; ...
    'fid',      1 };
  [got,opt] = parseopt( cOptDef, varargin{:} );
  fid       = opt.fid;

% Grid generation.
  nx       = round( opt.len/opt.hmax );
  fea.grid = linegrid( nx, 0, opt.len );


% Problem definition.
  u = opt.dvname;
  switch( opt.icase )
    case 1
      seqn = [u,''' - 1/2*',u,'x_x - ',u,'x_t + ',u,'_t = (x-t)^5 - 10*(x-t)^4 - 10*(x-t)^3'];
    case 2
      seqn = [u,''' - 1/2*x^2*',u,'x_x - x*',u,'x_t + ',u,'_t = (x-t)^5 - 5*(x-t)^4 - 5*x*(x-t)^4 -
  end
  refsol = '(x-t).^5';
  init_u = 'x^5';


% Set up problem struct.
  fea.sdim  = { 'x' };
  fea = addphys( fea, @customeqn );
  fea.phys.ce.dvar = { u };
  fea.phys.ce.eqn.seqn = seqn;
  fea.phys.ce.sfun = { opt.sfun };


% Dirichlet BCs for the left and right boundaries.
  fea.phys.ce.bdr.coef{1,5} = { 1 1 };
  fea.phys.ce.bdr.coef{1,7} = { '-t^5' ['(',num2str(opt.len),'-t)^5'] };
```

```
% Check and parse problem struct.
 fea = parsephys( fea );
 fea = parseprob( fea );


% Call to time-dependent solver.
 [fea.sol.u,tlist] = solvetime( fea, 'fid',     fid, ...
                                     'tmax',    opt.tmax, ...
                                     'init',    init_u, ...
                                     'icub',    6, ...
                                     'ischeme', opt.ischeme, ...
                                     'tstep',   opt.tmax/100);

% Postprocessing.
 fea.sol.t = tlist(end);
 fea.sol.u = fea.sol.u(:,end);
 refsol    = strrep( refsol, 't', num2str(fea.sol.t) );
 if( opt.iplot>0 )

   figure
   postplot( fea, 'surfexpr', u, 'axequal', 'off', 'linewidth', 2 )
   title( ['Solution at time ',num2str(fea.sol.t)] )
   hold on
   x = linspace(0,opt.len,25);
   u_ref = eval(refsol);
   plot( x, u_ref, 'r--' )
 end


% Error checking.
 err = evalexpr( ['abs(',refsol,'-',u,')'], linspace(0,opt.len,10), fea );
 err = norm(err);

 if( ~isempty(fid) )
   fprintf(fid,'\nL2 Error: %f\n',err)
   fprintf(fid,'\n\n')
 end

 out.err  = err;
 out.pass = out.err<opt.tol;
 if ( nargout==0 )
   clear fea out
 end
```

11.90 examples/ex_euler_beam1

EX_EULER_BEAM1 1D Euler-Bernoulli beam model example.

[FEA, OUT] =

EX_EULER_BEAM1(VARARGIN) 1D Euler-Bernoulli beam model example. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
L           scalar {2}             Beam length
E           scalar {3}             Elastic modulus
I           expression {4}         Cross section moment of intertia
q           expression {5}         Beam force
nx          scalar {6}             Number of grid cells
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'L',        2;
             'E',        3;
             'I',        4;
             'q',        5;
             'nx'        6;
             'iplot',    1;
             'tol',      1e-2;
             'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;


% Grid generation.
 fea.sdim = {'x'};
 fea.grid = linegrid( opt.nx, 0, opt.L );


% Problem and equation definitions.
 fea = addphys( fea, @eulerbeam );
 fea.phys.eb.eqn.coef{3,end}  = { opt.E };
 fea.phys.eb.eqn.coef{4,end}  = { opt.I };
 fea.phys.eb.eqn.coef{5,end}  = { opt.q };
 fea.phys.eb.bdr.coef{1,5}{2} = 0;
 fea = parsephys( fea );
```

```
% Coefficients and equation/postprocessing expressions.
 fea.expr = { 'L',  opt.L ;
              'M',  fea.phys.eb.eqn.vars{3,2} };


% Parse and solve problem.
 fea = parseprob( fea );
 fea.sol.u = solvestat( fea, 'icub', 3, 'fid', opt.fid );


% Postprocessing.
 if( opt.iplot )
   figure
   subplot(3,1,1), hold on
   postplot( fea, 'surfexpr', 'E_eb*I_eb*v/(q_eb*L^4)', 'linewidth', 2 )
   postplot( fea, 'surfexpr', 'x^2*(6*L^2-4*L*x+x^2)/24/L^4', 'color', 'r', 'linestyle', ':' )
   title( 'E_eb*I_eb*v(x)/(q_eb*L^4)' )
   axis normal, grid on

   subplot(3,1,2), hold on
   postplot( fea, 'surfexpr', 'E_eb*I_eb*vx/(q_eb*L^3)', 'linewidth', 2 )
   postplot( fea, 'surfexpr', 'x*(3*L^2-3*L*x+x^2)/6/L^3', 'color', 'r', 'linestyle', ':' )
   title( 'E_eb*I_eb*theta(x)/(q_eb*L^3)' )
   axis normal, grid on

   subplot(3,1,3), hold on
   postplot( fea, 'surfexpr', 'M/(q_eb*L^2)', 'linewidth', 2 )
   postplot( fea, 'surfexpr', '-1/2*(L-x)^2/L^2', 'color', 'r', 'linestyle', ':' )
   axis normal, grid on
   title( 'M(x)/(q_eb*L^2)' )
 end


% Error checking.
 err_v  = evalexpr( 'abs(v-q_eb*x^2*(6*L^2-4*L*x+x^2)/(24*E_eb*I_eb))', ...
                    linspace(0,opt.L,3*opt.nx), fea );
 err_th = evalexpr( 'abs(vx-q_eb*x*(3*L^2-3*L*x+x^2)/(6*E_eb*I_eb))', ...
                    linspace(0,opt.L,3*opt.nx), fea );
 err_M  = evalexpr( 'abs(vxx-q_eb/2*(L-x)^2/(E_eb*I_eb))', ...
                    linspace(0,opt.L,3*opt.nx), fea );
 err = norm([err_v;err_th;err_M]);


 out.err  = err;
 out.pass = out.err<opt.tol;
 if ( nargout==0 )
   clear fea out
 end
```

11.91 examples/ex_euler_beam2

EX_EULER_BEAM2 1D Euler-Bernoulli beam vibration example.

[FEA, OUT] =

EX_EULER_BEAM2(VARARGIN) 1D Euler-Bernoulli beam model example for. calculating vibration modes and eigenfrequencies. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
L           scalar {1}             Beam length
E           scalar {1}             Elastic modulus
I           expression {1}         Cross section moment of intertia
rho         expression {1}         Material density
A           expression {1}         Cross section area
nx          scalar {100}           Number of grid cells
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'L',        1;
             'E',        1;
             'I',        1;
             'rho',      1;
             'A',        1;
             'nx'        100;
             'iplot',    1;
             'tol',      1e-3;
             'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;


% Grid generation.
 fea.sdim = {'x'};
 fea.grid = linegrid( opt.nx, 0, opt.L );


% Problem and equation definitions.
% Problem and equation definitions.
```

```
 fea = addphys( fea, @eulerbeam );
 fea.phys.eb.eqn.coef{1,end}  = { opt.rho };
 fea.phys.eb.eqn.coef{2,end}  = { opt.A };
 fea.phys.eb.eqn.coef{3,end}  = { opt.E };
 fea.phys.eb.eqn.coef{4,end}  = { opt.I };
 fea.phys.eb.bdr.coef{1,5}{2} = 0;
 fea = parsephys( fea );


% Coefficients and equation/postprocessing expressions.
 fea.expr = { 'L',    opt.L ;
              'E',    opt.E ;
              'I',    opt.I ;
              'rho',  opt.rho ;
              'A',    opt.A };


% Parse and assemble matrices.
 fea = parseprob( fea );
 [M,A] = assembleprob( fea, 'f_m', 1, 'f_a', 1, 'f_sparse', 1, 'icub', 4 );


% Eliminate Dirichlet boundary conditions from matrices.
 ix_b = ones(size(M,1),1);
 [tmp,ix_b] = bdrsetd( [], ix_b, fea );
 ix_rem = find(ix_b==0);
 M(:,ix_rem) = [];
 A(:,ix_rem) = [];
 M = M';
 A = A';
 M(:,ix_rem) = [];
 A(:,ix_rem) = [];
 M = M';
 A = A';


% Solve for eigenvalues and eigenvectors.
 n_eigs = 6;
 [V,D] = eigs( A, M, n_eigs, 0 );
 efq = sqrt(diag(D))/(2*pi);


% Postprocessing.
 if( opt.iplot )
   figure, hold on, grid on
   cols = {'b' 'r' 'y' 'g' 'c' 'm'};
   x = fea.grid.p;
   ind_dof_v = find(ix_b(1:length(x)));
   for i=1:4
     y = zeros(size(x));
     y(ind_dof_v) = V(1:length(ind_dof_v),n_eigs-i+1);
     plot( x, y, 'color', cols{mod(i-1,6)+1}, 'linewidth', 2 )
```

```
    end
    legend( {'Mode 1' 'Mode 2' 'Mode 3' 'Mode 4'} )
    title( 'Vibration modes' )
    xlabel( 'x' )
    ylabel( 'Displacement' )
  end

% Error checking.
  efq_ref = [0.5596 3.5069 9.8194 19.2421 31.8086 47.5166]';
  err = abs(sort(efq(1:6))-efq_ref)./efq_ref;


  out.efq  = sort(efq);
  out.err  = err;
  out.pass = all(out.err<opt.tol);
  if ( nargout==0 )
    clear fea out
  end
```

11.92 examples/ex_nonlinear_pde1

EX_NONLINEAR_PDE1 1D nonlinear PDE example.

[FEA, OUT] =

EX_NONLINEAR_PDE1(VARARGIN) Solves the equation (x∗u∗u_x)_x = 0 with Dirichlet boundary conditions u(x=x1)=u1 and u(x=x2)=u2. Compares against the analytical solution u(x) = sqrt(2∗c1∗log(x)+2∗
with c1 = -(u1^2-u2^2)/ (log(x2)-log(x1))/2 and c2 = (log(x2)∗u1^2-u2^2∗log(x1))/(log(x2)-log(x1))/2.
Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {1/10}          Grid cell size
sfun        string {sflag1}        Finite element shape function
u1          scalar                 Dirichlet boundary value at x = x1
u2          scalar                 Dirichlet boundary value at x = x2
x1          scalar                 Left most domain x-coordinate
x2          scalar                 Right most domain x-coordinate
ischeme     scalar {-1}            Solver scheme (<0 = stationary)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'hmax',     1/10;
   'sfun',     'sflag1';
   'u1',       1;
   'u2',       2;
   'x1',       3;
   'x2',       5;
   'ischeme', -1;
   'iplot',    1;
   'tol',      1e-3;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Grid generation.
 nx       = round( 1/opt.hmax );
 fea.grid = linegrid( nx, opt.x1, opt.x2 );

% Coefficients and expressions.
 fea.sdim  = { 'x' };
 fea.expr  = { 'u1'     opt.u1;
               'u2'     opt.u2;
               'x1'     opt.x1;
               'x2'     opt.x2;
               'c1'     '-(u1^2-u2^2)/(log(x2)-log(x1))/2';
               'c2'     '(log(x2)*u1^2-u2^2*log(x1))/(log(x2)-log(x1))/2';
               'u_ref'  'sqrt(2*c1*log(x)+2*c2)' };

% Physics mode and equation settings.
 fea = addphys( fea, @customeqn, { 'u' } );
 fea.phys.ce.sfun = { opt.sfun };
 fea.phys.ce.eqn.seqn = 'u'' - x*u*ux_x = 0';
 fea.phys.ce.eqn.coef = { 'u0_ce', 'u_0', 'Initial condition for u', { 'u1+(u2-u1)*(x-x1)/(x2-x1)'

% Boundary settings.
 fea.phys.ce.bdr.coef = { 'bcnd_ce', 'Dirichlet/Neumann boundary conditions',...
                          'Dirichlet/Neumann boundary conditions', ...
                          {'Dirichlet, r_u', 'Neumann, g_u' }, ...
                          { 1 1 }, [], { 'u1' 'u2' } };


% Parse and solve problem.
 fea = parsephys( fea );
 fea = parseprob( fea );
 if( opt.ischeme<1 )
   fea.sol.u = solvestat( fea, 'fid', fid, 'init', {'u0_ce'} );
 else
```

```
    fea.sol.u = solvetime( fea, 'fid', fid, 'init', {'u0_ce'}, 'ischeme', opt.ischeme );
  end


% Postprocessing.
  if( opt.iplot>0 )
    figure
    subplot(3,1,1)
    postplot( fea, 'surfexpr', 'u', 'axequal', 'off' )
    title('Solution u')
    subplot(3,1,2)
    postplot( fea, 'surfexpr', 'u_ref', 'axequal', 'off' )
    title('Exact solution')
    subplot(3,1,3)
    postplot( fea, 'surfexpr', 'abs(u_ref-u)', 'axequal', 'off', 'evalstyle', 'exact' )
    title('Error')
  end


% Error checking.
  xi  = [1/2; 1/2];
  err = evalexpr0( 'abs(u_ref-u)', xi, 1, 1:size(fea.grid.c,2), [], fea );
  ref = evalexpr0( 'u',            xi, 1, 1:size(fea.grid.c,2), [], fea );
  err = sqrt(sum(err.^2)/sum(ref.^2));

  if( ~isempty(fid) )
    fprintf(fid,'\nL2 Error: %f\n',err)
    fprintf(fid,'\n\n')
  end

  out.err  = err;
  out.pass = out.err<opt.tol;
  if( nargout==0 )
    clear fea out
  end
```

11.93 examples/ex_nonlinear_pde2

EX_NONLINEAR_PDE2 2D nonlinear PDE example.

[FEA, OUT] =

EX_NONLINEAR_PDE2(VARARGIN) Solves (x∗u∗u_x)_x + uy_y = 0 with Dirichlet boundary conditions u(x=x1)=u1 and u(x=x2)=u2. Compares against the analytical solution u(x) = sqrt(2∗c1∗log(x)+2∗c2) with c1 = -(u1^2-u2^2)/ (log(x2)-log(x1))/2 and c2 = (log(x2)∗u1^2-u2^2∗log(x1))/(log(x2)-log(x1))/2. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {1/10}          Grid cell size
sfun        string {sflag1}        Finite element shape function
u1          scalar                 Dirichlet boundary value at x = x1
u2          scalar                 Dirichlet boundary value at x = x2
x1          scalar                 Left most domain x-coordinate
x2          scalar                 Right most domain x-coordinate
ischeme     scalar {-1}            Solver scheme (<0 = stationary)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'hmax',     1/10;
   'sfun',     'sflag1';
   'u1',       1;
   'u2',       2;
   'x1',       3;
   'x2',       5;
   'ischeme', -1;
   'iplot',    1;
   'tol',      1e-3;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Grid generation.
 nx       = round( 1/opt.hmax );
 fea.grid = rectgrid( nx, nx, [opt.x1 opt.x2;0 1] );

% Coefficients and expressions.
 fea.sdim  = { 'x' 'y' };
 fea.expr  = { 'u1'     opt.u1;
               'u2'     opt.u2;
               'x1'     opt.x1;
               'x2'     opt.x2;
               'c1'     '-(u1^2-u2^2)/(log(x2)-log(x1))/2';
               'c2'     '(log(x2)*u1^2-u2^2*log(x1))/(log(x2)-log(x1))/2';
               'u_ref'  'sqrt(2*c1*log(x)+2*c2)' };
```

```
% Physics mode and equation settings.
 fea = addphys( fea, @customeqn, { 'u' } );
 fea.phys.ce.sfun = { opt.sfun };
 fea.phys.ce.eqn.seqn = 'u'' - x*u*ux_x - uy_y = 0';
 fea.phys.ce.eqn.coef = { 'u0_ce', 'u_0', 'Initial condition for u', { 'u1+(u2-u1)*(x-x1)/(x2-x1)'

% Boundary settings.
 fea.phys.ce.bdr.coef = { 'bcnd_ce', 'Dirichlet/Neumann boundary conditions',...
                          'Dirichlet/Neumann boundary conditions', ...
                          {'Dirichlet, r_u', 'Neumann, g_u' }, ...
                          { 0 1 0 1 }, [], { 0 'u2' 0 'u1' } };


% Parse and solve problem.
 fea = parsephys( fea );
 fea = parseprob( fea );
 if( opt.ischeme<1 )
   fea.sol.u = solvestat( fea, 'fid', fid, 'init', {'u0_ce'} );
 else
   fea.sol.u = solvetime( fea, 'fid', fid, 'init', {'u0_ce'}, 'ischeme', opt.ischeme );
 end


% Postprocessing.
 if( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', 'u', 'surfhexpr', 'u' )
   title('Solution u')
   u_ref = evalexprp( 'u_ref', fea );
   hold on
   plot3( fea.grid.p(1,:), fea.grid.p(2,:), u_ref, 'kx' )
 end


% Error checking.
 err = evalexprp( 'abs(u_ref-u)', fea );
 ref = evalexprp( 'u',            fea );
 err = sqrt(sum(err.^2)/sum(ref.^2));

 if( ~isempty(fid) )
   fprintf(fid,'\nL2 Error: %f\n',err)
   fprintf(fid,'\n\n')
 end

 out.err  = err;
 out.pass = out.err<opt.tol;
 if( nargout==0 )
   clear fea out
 end
```

11.94 examples/ex_diffusion1

EX_DIFFUSION1 2D Diffusion equation example on a unit square.

[FEA, OUT] =

EX_DIFFUSION1(VARARGIN) Diffusion equation on a unit square with exact solutions. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
isol        scalar {1}             Exact solution
                    1                x*y
                    2                x^2-y^2
                    3                2*y/((1+x)^2+y^2)
                    4                2*y/((1+x)^2+y^2)
                    5                (sinh(pi*x)*sin(pi*y)+sinh(pi*y)*
                                     sin(pi*x))/sinh(pi)
cd          scalar {0.1}           Diffusion coefficient
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {1/40}          Max grid cell size
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'isol',     1; ...
   'cd',       0.1; ...
   'igrid',    0; ...
   'hmax',     1/40; ...
   'sfun',     'sflag1'; ...
   'iphys',    1; ...
   'iplot',    1; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;
 switch opt.isol
   case 1
```

```
      refsol = 'x*y';
    case 2
      refsol = 'x^2-y^2';
    case 3
      refsol = '2*y/((1+x)^2+y^2)';
    case 4
      refsol = 'exp(pi*x)*sin(pi*y)';   % pi can be substituted for any constant.
    case 5
      refsol = '(sinh(pi*x)*sin(pi*y)+sinh(pi*y)*sin(pi*x))/sinh(pi)';
  end


% Geometry definition.
  gobj = gobj_rectangle();
  fea.geom.objects = { gobj };


% Grid generation.
  switch opt.igrid
    case -1
      fea.grid = rectgrid(round(1/opt.hmax));
      fea.grid = quad2tri(fea.grid);
    case 0
      fea.grid = rectgrid(round(1/opt.hmax));
    case 1
      fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
  end
  n_bdr = max(fea.grid.b(3,:));           % Number of boundaries.


% Problem definition.
  fea.sdim  = { 'x' 'y' };                % Coordinate names.
  if ( opt.iphys==1 )

    fea = addphys(fea,@convectiondiffusion);   % Add convection and diffusion physics mode.
    fea.phys.cd.sfun = { opt.sfun };           % Set shape function.
    fea.phys.cd.eqn.coef{2,4}   = { opt.cd };  % Set diffusion coefficient.
    fea.phys.cd.bdr.sel         = [1 1 1 1];
    fea.phys.cd.bdr.coef{1,end} = repmat({refsol},1,n_bdr);   % Set Dirichlet boundary coefficient
    fea = parsephys(fea);                  % Check and parse physics modes.

  else

    fea.dvar  = { 'c' };                   % Dependent variable name.
    fea.sfun  = { opt.sfun  };             % Shape function.

% Define equation system.
    fea.eqn.a.form = { [2 3;2 3] };        % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
    fea.eqn.a.coef = { opt.cd };           % Diffusion coefficient used in assembling stiffness matr
```

```
    fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
    fea.eqn.f.coef = { 0 };               % Coefficient used in right hand side.

% Define boundary conditions.
    fea.bdr.d     = cell(1,n_bdr);
   [fea.bdr.d{:}] = deal(refsol);         % Assign reference solution to all boundaries (Dirichlet)

    fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid);   % Call to stationary solver.


% Postprocessing.
 s_err = ['abs(',refsol,'-c)'];
 if ( opt.iplot>0 )
   figure
   subplot(2,1,1)
   postplot(fea,'surfexpr','c')
   title('Solution c')
   subplot(2,1,2)
   postplot(fea,'surfexpr',s_err,'evalstyle','exact')
   title('Error')
 end


% Error checking.
 if ( size(fea.grid.c,1)==4 )
   xi = [0;0];
 else
   xi = [1/3;1/3;1/3];
 end
 err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
 ref = evalexpr0('c',xi,1,1:size(fea.grid.c,2),[],fea);
 err = sqrt(sum(err.^2)/sum(ref.^2));


 if( ~isempty(fid) )
   fprintf(fid,'\nL2 Error: %f\n',err)
   fprintf(fid,'\n\n')
 end


 out.err  = err;
 out.pass = out.err<0.01;
 if ( nargout==0 )
   clear fea out
 end
```

11.95 examples/ex_diffusion2

EX_DIFFUSION2 1D Time dependent diffusion equation examples.

[FEA, OUT] =

EX_DIFFUSION2(VARARGIN) 1D time dependent diffusion equation on a line with exact solutions. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
nu          scalar {1e-1}          Diffusion coefficient
icase       scalar {2}             Test case
hmax        scalar {1/25}          Max grid cell size
dt          scalar {0.1}           Time step size
ischeme     scalar {3}             Time stepping scheme
sfun        string {sflag1}        Shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'nu',       1e-1; ...
   'icase',    2; ...
   'hmax',     1/25; ...
   'dt'        0.1; ...
   'ischeme'   3; ...
   'sfun',     'sflag1'; ...
   'iplot',    1; ...
   'tol',      1e-2; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


 tmax = 1;
```

```
 switch( opt.icase )
   case 1
     refsol = '(1-x)*2.2+x*3.3';
   case 2
     n = 1;
     refsol = ['exp(-',num2str(opt.nu*n^2*pi^2),'*t)*sin(',num2str(n*pi),'*x)'];
 end


% Grid generation.
 fea.grid = linegrid( 1/opt.hmax, 0, 1 );


% Problem definition.
 fea.sdim  = { 'x' };
 fea = addphys( fea, @convectiondiffusion );
 fea.phys.cd.sfun = { opt.sfun };
 fea.phys.cd.eqn.coef{2,4} = { opt.nu };
 fea = parsephys(fea);


% Parse and solve problem.
 x = fea.grid.p';
 n = length(x);
 if( strcmp( opt.sfun,'sflag2' ) )
   x = [ x; (x(2:end)+x(1:end-1))/2 ];
 end
 t   = 0;
 u0  = eval( refsol );
 fea = parseprob( fea );
 if( opt.ischeme>0 )

   fea.bdr.d{1} = refsol;
   fea.bdr.d{2} = refsol;
   fea.bdr.n    = cell(1,2);
   [fea.sol.u,tlist] = solvetime( fea, 'fid', fid, 'init', u0, 'ischeme', opt.ischeme, 'tstep', op

 else

   [M,A,f] = assembleprob( fea, 'f_m', 1, 'imass', 1, 'f_a', 1, 'f_f', 1, 'f_sparse', 1 );
   M = spdiags( full(sum(M')'), 0, size(M,1), size(M,1) );
   fea.sol.u = u0;
   dt = opt.dt;
   it = 0;
   tlist = 0;
   if( opt.ischeme==-1 )       % Crank-Nicolson.
     C = [ M + dt/2*A ];
   elseif( opt.ischeme==-2 )   % Backward Euler.
     C = [ M + dt*A ];
   end
   C(1,:) = 0;
   C(1,1) = 1;
```

```
    C(n,:) = 0;
    C(n,n) = 1;
    while 1
      t  = t  + dt;
      it = it + 1;

      u_r = eval( refsol );
      if( opt.ischeme==-1 )       % Crank-Nicolson.
        b = [ [ M - dt/2*A ]*u0 + dt*f ];
      elseif( opt.ischeme==-2 )   % Backward Euler.
        b = [ M*u0 + dt*f ];
      end
      b(1) = u_r(1);
      b(n) = u_r(n);

      u1 = C\b;

      if( t>=tmax )
        break
      end

      err = norm( u1 - u_r )/norm( u_r );
      errnm(it) = err;
      if( ~isempty(fid) )
        fprintf( fid, 'Time = %f, error norm = %d\n', t, err );
      end

      fea.sol.u = [ fea.sol.u u1 ];
      tlist = [ tlist t ];
      u0 = u1;
    end

  end


% Postprocessing.
  if( opt.iplot>0 )
    figure;
    if( opt.iplot>1 )
      i_sol_list = 1:numel(tlist);
    else
      i_sol_list = numel(tlist);
    end
    [~,ix] = sort( x );
    for i_sol=i_sol_list
      t = tlist(i_sol);
      clf
      postplot( fea, 'surfexpr', 'c', 'solnum', i_sol );
      hold on
      u_r = eval( refsol );
      plot( sort(x), u_r(ix), 'r--' );
      title( ['Solution at time ',num2str(t)])
```

```
      xlabel( 'x' )
      drawnow
    end
  end


% Error checking.
  for i_sol=1:numel(tlist)
    u_i = fea.sol.u(:,i_sol);
    t   = tlist(i_sol);
    u_r = eval( refsol );
    errnm(i_sol) = norm( u_i - u_r )/norm( u_r );
  end
  out.err  = errnm;
  out.pass = all( errnm<opt.tol );


  if ( nargout==0 )
    clear fea out
  end
```

11.96 examples/ex_heattransfer1

EX_HEATTRANSFER1 2D ceramic strip with radiation and convection.

[FEA, OUT] =

EX_HEATTRANSFER1(VARARGIN) 2D heat transfer of a ceramic strip with both radiation and convection on the top boundary.

```
  _                 q = h*(T_inf-T) + epsilon*sigma*(T_inf^4-T^4)
  ^              +-------------------+
  |              |                   |
  |              |                   |
0.01m  T=900C    |                   |  T=900C
  |              |                   |
  |              |                   |
  v              +-------------------+
                      dt/dn = 0
                 |<---- 0.02m ----->|
```

The ceramic has a thermal conductivity of 50 W/mC and the sides are fixed at a temperature of 900C while the bottom boundary is insulated. The surrounding temperature is 50C. The top boundary is exposed to both natural convection (with a film coefficient h=50W/m^2K) and

radiation (with emissivity epsilon=0.7 and the Stefan-Boltzmann 5.669e-8 W/m^2K^4). The solution is sought at three points along the vertical symmetry line.

Reference

```
[1] Holman, J. P., Heat Transfer, Fifth Edition, New York: McGraw-Hill,
    1981, page 96, Example 3-8.
```

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {0.001}         Grid cell size
igrid       scalar {0}/1/2         Cell type (0=quadrilaterals, 1=triangles,
sfun        string {sflag1}        Finite element shape function
iplot       scalar {1}/0           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'hmax',     0.001;
             'igrid',    0;
             'sfun',     'sflag1';
             'iplot',    1;
             'tol',      0.01;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});


% Geometry definition.
 gobj = gobj_rectangle( 0, 0.02, 0, 0.01 );
 fea.geom.objects = { gobj };


% Grid generation.
 switch opt.igrid
   case 0
```

```
      fea.grid = rectgrid( round(0.02/opt.hmax), round(0.01/opt.hmax), [0 0.02;0 0.01] );
    case 1
      fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', opt.fid );
    case 2
      fea.grid = rectgrid( round(0.02/opt.hmax), round(0.01/opt.hmax), [0 0.02;0 0.01] );
      fea.grid = quad2tri( fea.grid, 1 );
  end


% Problem definition.
  fea.sdim  = { 'x', 'y' };             % Space coordinate name.
  fea = addphys( fea, @heattransfer );  % Add heat transfer physics mode.
  fea.phys.ht.sfun = { opt.sfun };      % Set shape function.

% Equation coefficients.
  fea.phys.ht.eqn.coef{3,end} = 3;      % Thermal conductivity.

% Boundary conditions.
  fea.phys.ht.bdr.sel = [3 1 4 1];
  fea.phys.ht.bdr.coef{1,end}   = { [] 900+273 [] 900+273 };
  fea.phys.ht.bdr.coef{4,end}{3}{2} = 50;
  fea.phys.ht.bdr.coef{4,end}{3}{3} = 50+273;
  fea.phys.ht.bdr.coef{4,end}{3}{4} = 0.7*5.669e-8;
  fea.phys.ht.bdr.coef{4,end}{3}{5} = 50+273;


% Parse physics modes and problem stuct.
  fea = parsephys(fea);
  fea = parseprob(fea);


% Compute solution.
  fea.sol.u = solvestat( fea, 'fid', opt.fid, 'init', {'T0_ht'} );


% Postprocessing.
  if( opt.iplot>0 )
    postplot( fea, 'surfexpr', 'T', 'isoexpr', 'T' )
    title('Temperature, T')
  end


% Error checking.
  T2_sol = evalexpr( 'T', [0.01;0.01], fea );
  T2_ref = 984;
  T5_sol = evalexpr( 'T', [0.01;0.005], fea );
  T5_ref = 1064;
  T8_sol = evalexpr( 'T', [0.01;0], fea );
  T8_ref = 1088;
  out.err  = abs([T2_sol-T2_ref T5_sol-T5_ref T8_sol-T8_ref])./[T2_ref T5_ref T8_ref];
  out.pass = all(out.err<opt.tol);
```

```
 if( nargout==0 )
   clear fea out
 end
```

11.97 examples/ex_heattransfer2

EX_HEATTRANSFER2 1D Stationary heat transfer with radiation.

[FEA, OUT] =

EX_HEATTRANSFER2(VARARGIN) NAFEMS T2 benchmark example for heat transfer with radiation
[1]. The left end of a 0.1 m rod is held at a temperature of 1000 K while the right end is radiating with an emissivity, em = 0.98, and Stefan-Bolzmann constant, sigma = 5.67e-8 Wm^2/K^4.

```
       +---------- L=0.1m ----------+ T(0.1)?
   T=1000K                  q_n = em*sigma*(T_amb^4-T^4)
```

The rod is made of iron with density 7850 kg/m^3, heat capacity 460 J/kgK, and thermal conductivity 55.563 W/mK. The steady state temperature at the left end is sought when the surrounding ambient temperature is 300 K.

Reference


```
   [1] The Standard NAFEMS Benchmarks,
       The National Agency for Finite Element Standards, UK, 1990.
```


Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {0.02}          Grid cell size
sfun        string {sflag1}        Finite element shape function
istat       scalar {1}/0           Use stationary (=1), or time dependent solver
iplot       scalar {1}/0           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
```

```
fea         struct                  Problem definition struct
out         struct                  Output stuct
```

Code listing

```
 cOptDef = { 'hmax',     0.02;
             'sfun',     'sflag1';
             'istat',    1;
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});


% Grid generation.
 L        = 0.1;
 nx       = round(L/opt.hmax);
 fea.grid = linegrid( nx, 0, L );


% Problem definition.
 fea.sdim  = { 'x' };                       % Space coordinate name.
 fea = addphys( fea, @heattransfer );       % Add heat transfer physics mode.
 fea.phys.ht.sfun = { opt.sfun };           % Set shape function.

% Equation coefficients.
 fea.phys.ht.eqn.coef{1,end} = 7850;        % Density
 fea.phys.ht.eqn.coef{2,end} =  460;        % Heat capacity.
 fea.phys.ht.eqn.coef{3,end} =   55.563;    % Thermal conductivity.
 fea.phys.ht.eqn.coef{6,end} = { 1000 };    % Initial temperature.

% Boundary conditions.
 fea.phys.ht.bdr.sel = [ 1 4 ];
 fea.phys.ht.bdr.coef{1,end} = { 1000 [] };
 fea.phys.ht.bdr.coef{4,end}{2}{4} = '0.98*5.67e-8';
 fea.phys.ht.bdr.coef{4,end}{2}{5} = 300;


% Parse physics modes and problem stuct.
 fea = parsephys(fea);
 fea = parseprob(fea);


% Compute solution.
 if( opt.istat )
   fea.sol.u = solvestat( fea, 'fid', opt.fid, 'init', {'T0_ht'} );
 else
   [fea.sol.u, tlist] = solvetime( fea, 'fid', opt.fid, 'init', {'T0_ht'}, ...
```

```
                                        'tmax', 1000, 'tstep', 10 );
 end


% Postprocessing.
 if( opt.iplot>0 )
   postplot( fea, 'surfexpr', 'T', 'axequal', 0 )
   title('Temperature')
   xlabel('x')
   ylabel('T')
 end


% Error checking.
 T_sol = evalexpr( 'T', 0.1, fea );
 T_ref = 926.97;
 out.err  = abs(T_sol-T_ref)/T_ref;
 out.pass = out.err<6e-4;


 if( nargout==0 )
   clear fea out
 end
```

11.98 examples/ex_heattransfer3

EX_HEATTRANSFER3 1D Transient heat conduction.

[FEA, OUT] =

EX_HEATTRANSFER3(VARARGIN) NAFEMS T3 benchmark example for one-dimensional transient heat conduction
[1]. A 10 cm thick steel plate is assumed to have one surface exposed to a varying temperature, T = 100∗sin(pi∗t/40), while the temperature at the other side is fixed, T = 0. This problem can be seen as one dimensional along the axis aligned with the thickness.

```
     +---T(0.02)?---- L=0.02m ------------+
T=100*sin(pi*t/40)                        T=0
```

The temperature at x = 0.02 m is sought at time t = 36. The material parameters of the plate are, density 7200 kg/m^3, heat capacity 440.5 J/kgK, and thermal conductivity 35 W/mK.

Reference


```
   [1] The Standard NAFEMS Benchmarks,
       The National Agency for Finite Element Standards, UK, 1990.
```


Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {0.005}         Grid cell size
sfun        string {sflag1}        Finite element shape function
iplot       scalar {1}/0           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'hmax',     0.005;
             'sfun',     'sflag1';
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});


% Grid generation.
 L        = 0.1;
 nx       = round(L/opt.hmax);
 fea.grid = linegrid( nx, 0, L );


% Problem definition.
 fea.sdim  = { 'x' };                  % Space coordinate name.
 fea = addphys( fea, @heattransfer );  % Add heat transfer physics mode.
 fea.phys.ht.sfun = { opt.sfun };      % Set shape function.

% Equation coefficients.
 fea.phys.ht.eqn.coef{1,end} = 7200;   % Density
 fea.phys.ht.eqn.coef{2,end} =  440.5; % Heat capacity.
 fea.phys.ht.eqn.coef{3,end} =   35;   % Thermal conductivity.
```

```
% Boundary conditions.
 fea.phys.ht.bdr.sel = [ 1 1 ];
 fea.phys.ht.bdr.coef{1,end} = { '100*sin(pi*t/40)' 0 };


% Parse physics modes and problem stuct.
 fea = parsephys(fea);
 fea = parseprob(fea);


% Compute solution.
 [fea.sol.u, tlist] = solvetime( fea, 'fid', opt.fid, 'tmax', 32, 'tstep', 0.05 );


% Postprocessing.
 if( opt.iplot>0 )
   figure
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'T', 'axequal', 'off' )
   title('Temperature distribution at time t = 32 s')
   xlabel('x')
   ylabel('T')

   subplot(1,2,2)
   for isol=1:numel(tlist)
     T(isol) = evalexpr( 'T', 0.02, fea, isol );
   end
   plot(tlist,T)
   title('Temperature at x = 0.02 m')
   xlabel('time')
   ylabel('T')
 end


% Error checking.
 T_sol = evalexpr( 'T', 0.02, fea );
 T_ref = 36.6;
 out.err  = abs(T_sol-T_ref)/T_ref;
 out.pass = out.err<1e-2;


 if( nargout==0 )
   clear fea out
 end
```

11.99 examples/ex_heattransfer4

EX_HEATTRANSFER4 2D Heat transfer with convective cooling.

[FEA, OUT] =

EX_HEATTRANSFER4(VARARGIN) NAFEMS T4 benchmark example for two dimensional heat transfer with convective heat flux boundary conditions.

```
 _            q_n=h*(T_amb-T)
 ^         +---------+
 |         |         |
 |  q_n=0  |         | q_n=h*(T_amb-T)
1m         |         |
 |         |   T(0.6,0.2)?
 |         |         |
 v         +---------+
              T=100

           |<-0.6m->|
```

A 0.6 by 1 m iron plate, with density 7850 kg/m^3, heat capacity 460 J/kgC, and thermal conductivity 52 W/mC, is prescribed a fixed temperature of T = 100 C at the bottom edge. The left side is insulated, and the right and top boundaries exposed to convective cooling with a heat transfer coefficient h = 750 W/m^2K. The steady state temperature at the point (0.6,0.2) is sought when the surrounding ambient temperature is T_amb = 0 C.

Reference

```
   [1] Cameron AD, Casey JA, Simpson GB. Benchmark Tests for Thermal Analysis,
       The National Agency for Finite Element Standards, UK, 1986.
```

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {0.025}         Grid cell size
igrid       scalar {0}/1/2         Cell type (0=quadrilaterals, 1=triangles,
                                   2=triangles converted from quadrilaterals)
sfun        string {sflag1}        Finite element shape function
istat       scalar {1}/0           Use stationary (=1), or time dependent solver
iplot       scalar {1}/1           Plot solution (=1)
```

```
                                                                                .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'hmax',     0.025;
             'igrid',    0;
             'sfun',     'sflag1';
             'istat',    1;
             'iplot',    1;
             'tol',      1e-2;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});


% Geometry definition.
 gobj = gobj_rectangle( 0, 0.6, 0, 1 );
 fea.geom.objects = { gobj };


% Grid generation.
 switch opt.igrid
   case 0
     fea.grid = rectgrid( round(0.6/opt.hmax), round(1/opt.hmax), [0 0.6;0 1] );
   case 1
     fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', opt.fid );
   case 2
     fea.grid = rectgrid( round(0.6/opt.hmax), round(1/opt.hmax), [0 0.6;0 1] );
     fea.grid = quad2tri( fea.grid, 1 );
 end


% Problem definition.
 fea.sdim  = { 'x', 'y' };              % Space coordinate name.
 fea = addphys( fea, @heattransfer );  % Add heat transfer physics mode.
 fea.phys.ht.sfun = { opt.sfun };      % Set shape function.

% Equation coefficients.
 fea.phys.ht.eqn.coef{1,end} = 7850;   % Density
 fea.phys.ht.eqn.coef{2,end} =  460;   % Heat capacity.
 fea.phys.ht.eqn.coef{3,end} =   52;   % Thermal conductivity.
 fea.phys.ht.eqn.coef{7,end} = { 0 };  % Initial temperature.

% Boundary conditions.
```

```
 fea.phys.ht.bdr.sel = [1 4 4 3];
 fea.phys.ht.bdr.coef{1,end}   = { 100 [] [] [] };
 fea.phys.ht.bdr.coef{4,end}{2}{2} = 750;
 fea.phys.ht.bdr.coef{4,end}{3}{2} = 750;


% Parse physics modes and problem stuct.
 fea = parsephys(fea);
 fea = parseprob(fea);


% Compute solution.
 if( opt.istat )
   fea.sol.u = solvestat( fea, 'fid', opt.fid, 'init', {'T0_ht'} );
 else
   [fea.sol.u, tlist] = solvetime( fea, 'fid', opt.fid, 'init', {'T0_ht'}, ...
                                        'tmax', 20000, 'tstep', 100, 'toldef', 1e-4, 'maxnit', 5 )
 end


% Postprocessing.
 if( opt.iplot>0 )
   postplot( fea, 'surfexpr', 'T', 'isoexpr', 'T' )
   title('Temperature, T')
 end


% Error checking.
 T_sol = evalexpr( 'T', [0.6;0.2], fea );
 T_ref = 18.3;
 out.err  = abs(T_sol-T_ref)/T_ref;
 out.pass = out.err<opt.tol;


 if( nargout==0 )
   clear fea out
 end
```

11.100 examples/ex_heattransfer5

EX_HEATTRANSFER5 2D Transient cooling shrink fitting example.

[FEA, OUT] =

EX_HEATTRANSFER5(VARARGIN) This example models transient cooling for shrink fitting a two part assembly. A tungsten rod heated to 84 C is inserted into a -10 C chilled steel frame part. The time when the maximum temperature has cooled to 70 C should be determined. The

assembly is cooled due to convection through a surrounding medium kept at 17 C and a heat transfer coefficient of 750 W/m^2K
[1].

Reference


```
[1] Krysl P. A Pragmatic Introduction to the Finite Element Method
    for Thermal and Stress Analysis. Pressure Cooker Press, USA, 2005.
```


Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {0.0025}        Grid cell size
sfun        string {sflag1}        Finite element shape function
iplot       scalar {1}/0           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'hmax',     0.005;
             'sfun',     'sflag1';
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});


% Geometry definition.
 r1 = gobj_rectangle( 0, 0.11, 0, 0.12,  'R1' );
 c1 = gobj_circle( [ 0.065 0 ],   0.015, 'C1' );
 c2 = gobj_circle( [ 0.11 0.12 ], 0.035, 'C2' );
 c3 = gobj_circle( [ 0 0.06 ],    0.025, 'C3' );

 r2 = gobj_rectangle( 0.065, 0.16, 0.05, 0.07, 'R2' );
 c4 = gobj_circle( [ 0.065 0.06 ], 0.01, 'C4' );

 fea.geom.objects = { r1 c1 c2 c3 r2 c4 };
```

```
 fea = geom_apply_formula( fea, 'R1-C1-C2-C3' );
 fea = geom_apply_formula( fea, 'R2+C4' );


% Grid generation.
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', opt.fid );


% Problem definition.
 fea.sdim  = { 'x', 'y' };             % Space coordinate names.
 fea = addphys( fea, @heattransfer );  % Add heat transfer physics mode.
 fea.phys.ht.sfun = { opt.sfun };      % Set shape function.

% Equation coefficients.
 rho_steel    =  7500;
 rho_tungsten = 19000;
 cp_steel     =   470;
 cp_tungsten  =   134;
 k_steel      =    44;
 k_tungsten   =   163;
 fea.phys.ht.eqn.coef{1,end} = { rho_tungsten rho_steel rho_tungsten };   % Density
 fea.phys.ht.eqn.coef{2,end} = { cp_tungsten  cp_steel  cp_tungsten  };   % Heat capcity.
 fea.phys.ht.eqn.coef{3,end} = { k_tungsten   k_steel   k_tungsten   };   % Thermal conductivity.
 fea.phys.ht.eqn.coef{7,end} = { 84 -10 84 };                             % Initial temperature.

% Boundary conditions.
 n_bdr = max(fea.grid.b(3,:));
 fea.phys.ht.bdr.sel = 4*ones(1,n_bdr);
 for i_bdr=1:n_bdr
   fea.phys.ht.bdr.coef{4,end}{i_bdr}{2} = 'k_ht';
   fea.phys.ht.bdr.coef{4,end}{i_bdr}{3} = 17;
 end


% Parse physics modes and problem stuct.
 fea = parsephys(fea);
 fea = parseprob(fea);


% Compute solution.
 [fea.sol.u, tlist] = solvetime( fea, 'fid', opt.fid, 'tmax', 16, 'tstep', 0.25, 'tstop', 0, ...
                                      'init', {'T0_ht'}, 'maxnit', 2 );

% Check when max temperature < 70.
 for i=1:numel(tlist)
   T_min(i) = min(fea.sol.u(:,i));
   T_max(i) = max(fea.sol.u(:,i));
 end
 ix = find(T_max<70);
 i1 = ix(1);
 i2 = i1 - 1;
 s  = ( T_max(i2) - 70 )/( T_max(i2) - T_max(i1) );
```

```
 t_70 = tlist(i2) + s*( tlist(i1) - tlist(i2) );
 u_70 = fea.sol.u(:,i2) + s*( fea.sol.u(:,i1) - fea.sol.u(:,i2) );


% Postprocessing.
 if( opt.iplot>0 )
   figure
   fea_plot = fea;
   fea_plot.sol.u = u_70;
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'T', 'isoexpr', 'T', 'isolev', 20 )
   title(['Temperature distribution at t=',num2str(t_70)])

   subplot(1,2,2)
   plot( tlist, T_min, 'b-' )
   hold on
   plot( tlist, T_max, 'r-' )
   grid on
   title('Maximum and minimum temperatures')
   ylabel('Temperature [C]')
   xlabel('Time [s]')
 end


% Error checking.
 out.T_min = T_min;
 out.T_max = T_max;
 out.T_70  = t_70;
 out.pass = abs(t_70-13)<3.5;


 if( nargout==0 )
   clear fea out
 end
```

11.101 examples/ex_heattransfer6

EX_HEATTRANSFER6 2D axisymmetric heat conduction.

[FEA, OUT] =

EX_HEATTRANSFER6(VARARGIN) NAFEMS benchmark example for heating of a solid cylider with an internal hole.

```
  _              T=T_amb
  ^            +---------+
  |    q_n=0   |         |
```

```
  |            :          |
0.14m  q_n=5e5 |          | T=T_amb
  |            :          |
  |    q_n=0   |          |
  v            +----------+
                 T=T_amb
               r=0.02
               |<-0.08m->|
```

The geometry can be considered axisymmetric and the solid has a thermal conductivity of 52 W/mC, the middle part of the inside of the cylider is heated by 5e5 W/mK. The steady temperature at the point (0.04,0.04) is sought when the surrounding ambient temperature is T_amb = 0 C.

Reference

```
[1] Cameron AD, Casey JA, Simpson GB. Benchmark Tests for Thermal Analysis,
    The National Agency for Finite Element Standards, UK, 1986.
```

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {0.005}         Grid cell size
sfun        string {sflag1}        Finite element shape function
iplot       scalar {1}/0           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'hmax',     0.005;
             'sfun',     'sflag1';
             'iplot',    1;
             'tol',      1e-2;
             'fid',      1 };
```

```
 [got,opt] = parseopt(cOptDef,varargin{:});


% Geometry definition.
 gobj = gobj_polygon( [ 0.02 0.1 0.1  0.02 0.02 0.02 ;
                        0    0   0.14 0.14 0.1  0.04 ]' );
 fea.geom.objects = { gobj };


% Grid generation.
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', opt.fid );


% Problem definition.
 fea.sdim  = { 'r', 'z' };             % Space coordinate name.
 fea = addphys( fea, @heattransfer );  % Add heat transfer physics mode.
 fea.phys.ht.sfun = { opt.sfun };      % Set shape function.

% Equation coefficients.
 fea.phys.ht.eqn.seqn = '- r*k_ht*(Tr_r + Tz_z) = 0';

 fea.phys.ht.eqn.coef{3,end} =    52;       % Thermal conductivity.
 fea.phys.ht.eqn.coef{7,end} = { 273.15 };  % Initial temperature.

% Boundary conditions.
 fea.phys.ht.bdr.sel = [1 1 1 3 4 3];
 fea.phys.ht.bdr.coef{1,end}   = { 273.15 273.15 273.15 [] [] [] };
 fea.phys.ht.bdr.coef{4,end}{5}{1} = 'r*5e5';


% Parse physics modes and problem stuct.
 fea = parsephys(fea);
 fea = parseprob(fea);


% Compute solution.
 fea.sol.u = solvestat( fea, 'fid', opt.fid, 'init', {'T0_ht'} );


% Postprocessing.
 if( opt.iplot>0 )
   postplot( fea, 'surfexpr', 'T', 'isoexpr', 'T' )
   title('Temperature, T')
 end


% Error checking.
 T_sol = evalexpr( 'T', [0.04;0.04], fea );
 T_ref = 332.97;
 out.err  = abs(T_sol-T_ref)/T_ref;
 out.pass = out.err<opt.tol;
```

```
if( nargout==0 )
  clear fea out
end
```

11.102 examples/ex_heattransfer7

EX_HEATTRANSFER7 1D Transient heat diffusion with analytic solution.

[FEA, OUT] =

EX_HEATTRANSFER7(VARARGIN) Transient heat diffusion problem with analytic solution. A 1 m rod is kept at fixed temperature on one end and constant heat flux at the other as in the illustration.

```
       +---------- L=1m ----------+ T = 25
     q_n = 1       T(t=0) = 25
```

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------------
hmax        scalar {0.1}           Grid cell size
sfun        string {sflag1}        Finite element shape function
ischeme     scalar {2}/1/3         Time stepping scheme
tmax        scalar {0.2}           Maximum time
tstep       scalar {0.01}          Time step size
iplot       scalar {1}/0           Plot solution (=1)
                                                                                    .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
cOptDef = { 'hmax',     0.1;
            'sfun',     'sflag1';
            'ischeme',  2;
            'tmax',     0.2;
```

```
              'tstep',    0.01;
              'iplot',    1;
              'tol',      1e-3;
              'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});


% Grid generation.
 fea.grid = linegrid( round(1/opt.hmax), 0, 1 );


% Problem definition.
 fea.sdim  = { 'x' };                       % Space coordinate name.
 fea = addphys( fea, @heattransfer );       % Add heat transfer physics mode.
 fea.phys.ht.sfun = { opt.sfun };           % Set shape function.

% Equation coefficients.
 fea.phys.ht.eqn.coef{1,end} = 1;           % Density
 fea.phys.ht.eqn.coef{2,end} = 1;           % Heat capacity.
 fea.phys.ht.eqn.coef{3,end} = 1;           % Thermal conductivity.
 fea.phys.ht.eqn.coef{6,end} = { 25 };      % Initial temperature.

% Boundary conditions.
 fea.phys.ht.bdr.sel = [ 4 1 ];
 fea.phys.ht.bdr.coef{1,end} = { [] 25 };
 fea.phys.ht.bdr.coef{4,end}{1}{1} = -1;


% Parse physics modes and problem stuct.
 fea = parsephys(fea);
 fea = parseprob(fea);


% Compute solution.
 [fea.sol.u, tlist] = solvetime( fea, 'fid', opt.fid, 'init', {'T0_ht'}, 'ischeme', opt.ischeme, .
                                 'tmax', opt.tmax, 'tstep', opt.tstep );


% Postprocessing.
 T_ref = refsol( fea.grid.p', tlist(end) );
 if( opt.iplot>0 )
   postplot( fea, 'surfexpr', 'T', 'axequal', 0 )
   title(['Temperature at t=',num2str(tlist(end))])
   xlabel('x')
   ylabel('T')

   hold on
   plot( fea.grid.p, T_ref, 'r--' )
 end


% Error checking.
```

```
  T_sol = evalexpr( 'T', fea.grid.p, fea );
  out.err  = norm( abs(T_sol-T_ref)/T_ref );
  out.pass = out.err<opt.tol;


  if( nargout==0 )
    clear fea out
  end


% ----------------------------------
```

11.103 examples/ex_laplace1

EX_LAPLACE1 2D Laplace equation example on a unit square.

[FEA, OUT] =

EX_LAPLACE1(VARARGIN) Laplace equation on a unit square with exact solution 2∗y/((1+x)^2+y^2). Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {1/10}          Max grid cell size
refsol      string {2*y/((1+x)^2+y^2)}   Reference solution
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
  cOptDef = { ...
    'igrid',    0; ...
    'hmax',     0.1; ...
    'refsol',   '2*y/((1+x)^2+y^2)'; ...
```

```
    'sfun',     'sflag1'; ...
    'iphys',    1; ...
    'icub',     2; ...
    'iplot',    1; ...
    'fid',      1 };
  [got,opt] = parseopt(cOptDef,varargin{:});
  fid       = opt.fid;


% Geometry definition.
  gobj = gobj_rectangle();
  fea.geom.objects = { gobj };


% Grid generation.
  if ( opt.igrid==-1 )
    fea.grid = rectgrid(round(1/opt.hmax));
    fea.grid = quad2tri( fea.grid );
  elseif ( opt.igrid==0 )
    fea.grid = rectgrid(round(1/opt.hmax));
  else
    fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
  end
  n_bdr = max(fea.grid.b(3,:));           % Number of boundaries.


% Problem definition.
  fea.sdim  = { 'x' 'y' };                % Coordinate names.
  if ( opt.iphys==1 )

    fea = addphys(fea,@poisson);          % Add Poisson equation physics mode.
    fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
    fea.phys.poi.eqn.coef{3,4} = { 0 };   % Set source term coefficient to zero.
    fea.phys.poi.bdr.coef{1,end} = repmat({opt.refsol},1,n_bdr);   % Set Dirichlet boundary coeffic
    fea = parsephys(fea);                 % Check and parse physics modes.

  else

    fea.dvar  = { 'u' };                  % Dependent variable name.
    fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
    fea.eqn.a.form = { [2 3;2 3] };       % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
    fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.

    fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
    fea.eqn.f.coef = { 0 };               % Coefficient used in right hand side.

% Define boundary conditions.
    fea.bdr.d     = cell(1,n_bdr);
```

```
   [fea.bdr.d{:}] = deal(opt.refsol);     % Assign reference solution to all boundaries (Dirichlet)

    fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid,'icub',opt.icub);   % Call to stationary solver.


% Postprocessing.
 s_err = ['abs(',opt.refsol,'-u)'];
 if ( opt.iplot>0 )
   figure
   subplot(2,1,1)
   postplot(fea,'surfexpr','u')
   title('Solution u')
   subplot(2,1,2)
   postplot(fea,'surfexpr',s_err)
   title('Error')
 end


% Error checking.
 if ( size(fea.grid.c,1)==4 )
   xi = [0;0];
 else
   xi = [1/3;1/3;1/3];
 end
 err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
 ref = evalexpr0('u',xi,1,1:size(fea.grid.c,2),[],fea);
 err = sqrt(sum(err.^2)/sum(ref.^2));

 if( ~isempty(fid) )
   fprintf(fid,'\nL2 Error: %f\n',err)
   fprintf(fid,'\n\n')
 end

 out.err  = err;
 out.pass = out.err<0.01;
 if ( nargout==0 )
   clear fea out
 end
```

11.104 examples/ex_laplace2

EX_LAPLACE2 2D Laplace equation on a circle with nonzero boundary conditions.

[FEA, OUT] =

EX_LAPLACE2(VARARGIN) Laplace equation on a circle with source term 0, nonzero boundary conditions and exact solution u=r^3∗sin(3∗th). Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
radi        scalar {1}             Radius of circle
hmax        scalar {0.1}           Max grid cell size
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'radi',     1; ...
   'hmax',     0.1; ...
   'refsol',   'hypot(x,y)^3*sin(3*atan2(y,x))'; ...
   'sfun',     'sflag1'; ...
   'icub',     2; ...
   'iphys',    1; ...
   'iplot',    1; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 gobj = gobj_circle( [0 0], opt.radi );
 fea.geom.objects = { gobj };


% Grid generation.
 fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 n_bdr    = max(fea.grid.b(3,:));        % Number of boundaries.
```

```
% Problem definition.
 fea.sdim  = { 'x' 'y' };                % Coordinate names.
 if ( opt.iphys==1 )

   fea = addphys(fea,@poisson);          % Add Poisson equation physics mode.
   fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
   fea.phys.poi.eqn.coef{3,4} = { 0 };   % Set source term to zero.
   fea.phys.poi.bdr.coef{1,end} = repmat({opt.refsol},1,n_bdr);
   fea = parsephys(fea);                 % Check and parse physics modes.

 else

   fea.dvar  = { 'u' };                  % Dependent variable name.
   fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
   fea.eqn.a.form = { [2 3;2 3] };       % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
   fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.

   fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
   fea.eqn.f.coef = { 0 };               % Coefficient used in right hand side.

% Define boundary conditions.
   fea.bdr.d     = cell(1,n_bdr);
  [fea.bdr.d{:}] = deal(opt.refsol);     % Assign reference solution to all boundaries (Dirichlet)

   fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid,'icub',opt.icub);   % Call to stationary solver.


% Postprocessing.
 s_err = ['abs(',opt.refsol,'-u)'];
 if ( opt.iplot>0 )
   figure
   subplot(3,1,1)
   postplot(fea,'surfexpr','u','axequal','on')
   title('Solution u')
   subplot(3,1,2)
   postplot(fea,'surfexpr',opt.refsol,'axequal','on')
   title('Exact solution')
   subplot(3,1,3)
   postplot(fea,'surfexpr',s_err,'axequal','on')
```

```
    title('Error')
  end


 % Error checking.
  if ( size(fea.grid.c,1)==4 )
    xi = [0;0];
  else
    xi = [1/3;1/3;1/3];
  end
  err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
  ref = evalexpr0('u',xi,1,1:size(fea.grid.c,2),[],fea);
  err = sqrt(sum(err.^2)/sum(ref.^2));

  if( ~isempty(fid) )
    fprintf(fid,'\nL2 Error: %f\n',err)
    fprintf(fid,'\n\n')
  end


  out.err  = err;
  out.pass = out.err<5e-3;
  if ( nargout==0 )
    clear fea out
  end
```

11.105 examples/ex_linearelasticity1

EX_LINEARELASTICITY1 Example for solid stress-strain on a cube.

[FEA, OUT] =

EX_LINEARELASTICITY1(VARARGIN) Example to calculate displacements and stresses for a cube stretched in one direction.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {1}             Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
force       scalar {1}             Load force
idir        scalar 1/{2,3}         Stress direction
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.1}           Max grid cell size
sfun        string {sflag1}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
```

```
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'E',        1;
             'nu',       0.3;
             'force',    1;
             'idir',     1;
             'igrid',    0;
             'hmax',     0.1;
             'sfun',     'sflag1';
             'iplot',    1;
             'psecheck', 0;
             'tol',      sqrt(eps);
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 fea.geom.objects = { gobj_block() };
 fea.sdim = { 'x' 'y' 'z' };   % Coordinate names.


% Grid generation.
 switch opt.igrid
   case -1
     fea.grid = blockgrid( round(1/opt.hmax) );
     fea.grid = hex2tet( fea.grid );
   case 0
     fea.grid = blockgrid( round(1/opt.hmax) );
   case 1
     fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );
 end
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Boundary conditions.
 dtol     = opt.hmax;
 fixbdr   = findbdr( fea, [fea.sdim{opt.idir},'<',num2str(dtol)] );     % "Right" boundary number.
 forcebdr = findbdr( fea, [fea.sdim{opt.idir},'>',num2str(1-dtol)] );   % "Left" boundary number.
```

```
% Problem definition.
 fea = addphys(fea,@linearelasticity);
 fea.phys.el.eqn.coef{1,end} = { opt.nu };
 fea.phys.el.eqn.coef{2,end} = { opt.E  };
 fea.phys.el.sfun            = { opt.sfun opt.sfun opt.sfun };

% Fix first edge (set idir displacement to zero, Dirichlet BC).
 bctype = mat2cell( zeros(3,n_bdr), [1 1 1], ones(1,n_bdr) );
 bctype{opt.idir,fixbdr} = 1;
 fea.phys.el.bdr.coef{1,5} = bctype;

% Apply load to second opposite edge.
 bccoef = mat2cell( zeros(3,n_bdr), [1 1 1], ones(1,n_bdr) );
 bccoef{opt.idir,forcebdr} = opt.force;
 fea.phys.el.bdr.coef{1,end} = bccoef;


% Parse and solve problem.
 fea = parsephys( fea );
 fea = parseprob( fea );
 fea.sol.u = solvestat( fea, 'fid', fid, 'icub', 1+str2num(strrep(opt.sfun,'sflag','')) );


% Postprocessing.
 if( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', fea.dvar{opt.idir} )
   title( [fea.sdim{opt.idir},'-displacement'] )
 end


% Error checking.
 x = linspace(0,1,7);
 x = x(2:end-1)';
 [x,y,z] = ndgrid(x,x,x);
 p = [x(:) y(:) z(:)]';
 ui = evalexpr( fea.dvar{opt.idir}, p, fea );
 ui_ref = p(opt.idir,:)';
 out.ui_err = norm( ui - ui_ref );
 out.pserr  = [];
 out.peerr  = [];
 for i=1:6
   out.serr(i) = norm( abs( evalexpr( fea.phys.el.eqn.vars{5+i, end}, p, fea ) - (i==opt.idir) ) )
   out.eerr(i) = norm( abs( evalexpr( fea.phys.el.eqn.vars{14+i,end}, p, fea ) - (i==opt.idir) + 0
   if( opt.psecheck && i<=3 )
     out.pserr(i) = norm( abs( evalexpr( fea.phys.el.eqn.vars{11+i,end}, p, fea ) - (i==1) ) );
     out.peerr(i) = norm( abs( evalexpr( fea.phys.el.eqn.vars{20+i,end}, p, fea ) - (i==1) + 0.3*(
   end
 end
 out.serr = [ out.serr norm( abs( evalexpr( fea.phys.el.eqn.vars{1, end}, p, fea ) - 1 ) ) ];
 out.pass = all( [out.ui_err out.serr out.eerr out.pserr out.peerr] < opt.tol );
```

```
 if ( nargout==0 )
   clear fea out
 end
```

11.106 examples/ex_linearelasticity2

EX_LINEARELASTICITY2 Example for deflection of a bracket.

[FEA, OUT] =

EX_LINEARELASTICITY2(VARARGIN) Example to calculate displacements and stresses for a bracket with a circular hole.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {200e9}         Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
force       scalar {1e4}           Load force
l           scalar {0.2}           Length of bracket
t           scalar {0.02}          Thickness of bracket
r           scalar {0.08}          Radius of bracket hole
hmax        scalar {0.01}          Max grid cell size
sfun        string {sflag2}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'E',        200e9; ...
   'nu',       0.3; ...
   'force',    1e4; ...
   'l',        0.2; ...
   't',        0.02; ...
   'r',        0.08; ...
   'hmax',     0.01; ...
```

```
    'sfun',     'sflag2'; ...
    'iplot',    1; ...
    'tol',      0.05; ...
    'fid',      1 };
  [got,opt] = parseopt(cOptDef,varargin{:});
  fid       = opt.fid;


% Geometry definition.
  fea.sdim = { 'x' 'y' 'z' };   % Coordinate names.
  gobj1    = gobj_block( 0, opt.t, 0, opt.l, 0, opt.l, 'B1' );
  gobj2    = gobj_block( 0, opt.l, 0, opt.l, (opt.l-opt.t)/2, (opt.l+opt.t)/2, 'B2' );
  gobj3    = gobj_cylinder( [opt.l/2 opt.l/2 opt.l/2-opt.t], opt.r, 2*opt.t, 3, 'C1' );
  fea.geom.objects = { gobj1 gobj2 gobj3 };
  fea              = geom_apply_formula( fea, 'B1+B2-C1' );


% Grid generation.
  fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );


% Problem definition.
  fea = addphys(fea,@linearelasticity);
  fea.phys.el.eqn.coef{1,end} = { opt.nu };
  fea.phys.el.eqn.coef{2,end} = { opt.E  };
  fea.phys.el.sfun            = { opt.sfun opt.sfun opt.sfun };


% Boundary conditions.
  dtol = 2e-2;
  fixbdr   = findbdr( fea, ['x<=',num2str(dtol*1e-1)] );    % Right boundary number.
  forcebdr = findbdr( fea, ['x>=',num2str(opt.l-dtol)] );   % Left boundary number.

% Fix right edge (set zero Dirichlet BCs).
  n_bdr  = max(fea.grid.b(3,:));        % Number of boundaries.
  bctype = num2cell( zeros(3,n_bdr) );  % First set homogenous Neumann BCs everywhere.
  [bctype{:,fixbdr}] = deal( 1 );       % Set Dirchlet BCs for right boundary.
  fea.phys.el.bdr.coef{1,5} = bctype;

% Apply negative z-load to left edge.
  bccoef = num2cell( zeros(3,n_bdr) );
  bccoef{3,forcebdr} = -opt.force;
  fea.phys.el.bdr.coef{1,end} = bccoef;


% Parse and solve problem.
  fea       = parsephys( fea );
  fea       = parseprob( fea );
  warning('off')
  fea.sol.u = solvestat( fea, 'fid', fid );
  warning('on')
```

```
% Postprocessing.
 if ( opt.iplot>0 )
   DSCALE = 5000;

   subplot(2,2,1)
   postplot( fea, 'surfexpr', 'u' )
   title( 'x-displacement' )
   view([30 20])

   subplot(2,2,3)
   postplot( fea, 'surfexpr', 'v' )
   title( 'y-displacement' )
   view([30 20])

   subplot(2,2,2)
   postplot( fea, 'surfexpr', 'w' )
   title( 'z-displacement' )
   view([30 20])

   subplot(2,2,4)
   dp = zeros(size(fea.grid.p));
   for i=1:3
     dp(i,:) = DSCALE*evalexpr( fea.dvar{i}, fea.grid.p, fea );
   end
   fea_disp.grid   = fea.grid;
   fea_disp.grid.p = fea_disp.grid.p + dp;
   plotgrid( fea, 'facecolor', [.95 .95 .95], 'edgecolor', [.8 .8 1], ...
                  'selcells', selcells(fea,['x>',num2str(1.5*opt.t)]) )
   hold on
   plotgrid( fea_disp )
   title(['Displacement plot (at ',num2str(DSCALE),' times scale)'])
   view([30 20])

 end


% Error check.
 xdisp = fea.sol.u(fea.eqn.dofm{1}(:));
 ydisp = fea.sol.u(fea.eqn.dofm{2}(:)+fea.eqn.ndof(1));
 zdisp = fea.sol.u(fea.eqn.dofm{3}(:)+sum(fea.eqn.ndof(1:2)));
 xdisp = [ min(xdisp) max(xdisp) ];
 ydisp = [ min(ydisp) max(ydisp) ];
 zdisp = [ min(zdisp) max(zdisp) ];
 xdisp_ref = [ -8.971e-7 8.971e-7 ];
 ydisp_ref = [ -9.556e-8 9.605e-8 ];
 zdisp_ref = [ -1.09e-5  1.065e-8 ];
 svm_ref = [ 6.252756 1.779065e6 ];
 out.xdisp = xdisp;
 out.ydisp = ydisp;
 out.zdisp = zdisp;
 out.err   = abs(zdisp(1) - zdisp_ref(1))/abs(zdisp_ref(1));
```

```
 out.pass  = out.err<opt.tol;


 if ( nargout==0 )
   clear fea out
 end
```

11.107 examples/ex_linearelasticity3

EX_LINEARELASTICITY3 Parametric study for the bracket deflection model.

[FEA, OUT] =

EX_LINEARELASTICITY3(VARARGIN) Example to conduct parametric study of maximum stresses for the bracket deflection model.

```
Input       Value/{Default}           Description
-------------------------------------------------------------------------------
thickness   [0.035 0.03 0.025 0.02]   Plate thickness
loads       [1e4 2e4 1e5 1e6]         Applied loads.
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 thickness = [ 0.035 0.03 0.025 0.02 ];
 loads     = [ 1e4 2e4 1e5 1e6 ];

%% Starting new model.
 fea.sdim = { 'x', 'y', 'z' };
 fea = addphys( fea, @linearelasticity, { 'u', 'v', 'w' } );

 for i=1:length(thickness)

%% Geometry operations.
   fea.geom = struct;
   gobj = gobj_block( 0, 0.02, 0, 0.2, 0, 0.2, 'B1' );
   fea = geom_add_gobj( fea, gobj );
   gobj = gobj_block( 0, 0.2, 0, 0.2, 0.1-thickness(i)/2, 0.1+thickness(i)/2, 'B1' );
```

```
    fea = geom_add_gobj( fea, gobj );
    gobj = gobj_cylinder( [ 0.1, 0.1, 0.08 ], 0.08, 0.04, 3, 'C1' );
    fea = geom_add_gobj( fea, gobj );
    fea = geom_apply_formula( fea, 'B1+B2-C1' );

%% Grid operations.
    fea.grid = gridgen( fea, 'hmax', thickness(i)/2 );

    for j=1:length(loads)

%% Equation settings.
      fea.phys.el.dvar = { 'u', 'v', 'w' };
      fea.phys.el.sfun = { 'sflag1', 'sflag1', 'sflag1' };
      fea.phys.el.eqn.coef = { 'nu_el', 'nu', 'Poissons ratio', { '0.3' };
                               'E_el', 'E', 'Modulus of elasticity', { '200e9' };
                               'Fx_el', 'F_x', 'Body load in x-direction', { '0' };
                               'Fy_el', 'F_y', 'Body load in y-direction', { '0' };
                               'Fz_el', 'F_z', 'Body load in z-direction', { '0' };
                               'u0_el', 'u0', 'Initial condition for u', { '0' };
                               'v0_el', 'v0', 'Initial condition for v', { '0' };
                               'w0_el', 'w0', 'Initial condition for w', { '0' } };
      fea.phys.el.eqn.seqn = { 'u'' - ((E_el/((1+nu_el)*(1-2*nu_el))*(1-nu_el))*ux_x + (E_el/((1+nu
      fea.phys.el.eqn.vars = { 'von Mieses stress', 'sqrt(((E_el/((1+nu_el)*(1-2*nu_el))*(1-nu_el))
                               'x-displacement', 'u';
                               'y-displacement', 'v';
                               'z-displacement', 'w';
                               'Stress, x-component', '(E_el/((1+nu_el)*(1-2*nu_el))*(1-nu_el))*ux+
                               'Stress, y-component', '(E_el/((1+nu_el)*(1-2*nu_el))*nu_el)*ux+(E_e
                               'Stress, z-component', '(E_el/((1+nu_el)*(1-2*nu_el))*nu_el)*ux+(E_e
                               'Stress, xy-component', '(E_el/(1+nu_el)/2)*(uy+vx)';
                               'Stress, yz-component', '(E_el/(1+nu_el)/2)*(vz+wy)';
                               'Stress, zx-component', '(E_el/(1+nu_el)/2)*(uz+wx)';
                               'Displacement field', { 'u', 'v', 'w' } };

%% Constants and expressions.
      fea.expr = { 'load', loads(j) };

%% Boundary settings.
      n_bdr = max(fea.grid.b(3,:));
      ind_fixed = findbdr( fea, 'x<0.005' );
      ind_load  = findbdr( fea, 'x>0.19'  );
      fea.phys.el.bdr.sel = ones(1,n_bdr);

% Fix right edge (set zero Dirichlet BCs).
      n_bdr  = max(fea.grid.b(3,:));         % Number of boundaries.
      bctype = num2cell( zeros(3,n_bdr) );   % First set homogenous Neumann BCs everywhere.
      [bctype{:,ind_fixed}] = deal( 1 );     % Set Dirchlet BCs for right boundary.
      fea.phys.el.bdr.coef{1,5} = bctype;

% Apply negative z-load to left edge.
      bccoef = num2cell( zeros(3,n_bdr) );
      [bccoef{3,ind_load}] = deal( '-load' );
```

```
      fea.phys.el.bdr.coef{1,end} = bccoef;

%% Solver call.
      fea = parsephys( fea );
      fea = parseprob( fea );

      fea.sol.u = solvestat( fea );

%% Postprocessing.
      s_vm = fea.phys.el.eqn.vars{1,2};
      vm_stress = evalexpr( s_vm, fea.grid.p, fea );
      max_vm_stress(i,j) = max(vm_stress);

   end
 end


%% Visualization.
 [t,l] = meshgrid(thickness,loads);
 surf( t, l, max_vm_stress, log(max_vm_stress) )
 xlabel('Thickness')
 ylabel('Load')
 zlabel('Maximum stress')
 view(45,30)


 out = max_vm_stress;
 if ( nargout==0 )
   clear fea out
 end
```

11.108 examples/ex_linearelasticity4

EX_LINEARELASTICITY4 Stress calculation of an I-beam attached to two brackets.

[FEA, OUT] =

EX_LINEARELASTICITY4(VARARGIN) Example to calculate displacements and stresses for an I-beam suppored by two brackets with circular holes.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {200e9}         Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
force       scalar {1e5}           Load force
```

```
l           scalar {0.4}          Length of I-beam
ilev        scalar {2}            Grid regfinement level
sfun        string {sflag1}       Shape function for displacements
iplot       scalar 0/{1}          Plot solution (=1)
                                                                                .
Output      Value/(Size)          Description
--------------------------------------------------------------------------------
fea         struct                Problem definition struct
out         struct                Output struct
```

Code listing

```
 cOptDef = { ...
   'E',        200e9; ...
   'nu',       0.3; ...
   'force',    1e5; ...
   'l',        0.4; ...
   'ilev',     2; ...
   'sfun',     'sflag1'; ...
   'iplot',    1; ...
   'tol',      0.42; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 fea.sdim = { 'x' 'y' 'z' };   % Coordinate names.


% Grid generation.
 fea.grid = get_grid( opt.ilev );


% Problem definition.
 fea = addphys(fea,@linearelasticity);
 fea.phys.el.eqn.coef{1,end} = { opt.nu };
 fea.phys.el.eqn.coef{2,end} = { opt.E  };
 fea.phys.el.sfun            = { opt.sfun opt.sfun opt.sfun };


% Boundary conditions.
 dtol     = sqrt(eps);
 fixbdr   = findbdr( fea, ['(sqrt(x.^2+z.^2)<=0.03+sqrt(eps))&(z>=-sqrt(eps))'] );
 forcebdr = findbdr( fea, ['abs(y)>=0.2-sqrt(eps)'] );
```

```
% Fix edges (set zero Dirichlet BCs).
 n_bdr  = max(fea.grid.b(3,:));        % Number of boundaries.
 bctype = num2cell( zeros(3,n_bdr) );  % First set homogenous Neumann BCs everywhere.
 [bctype{:,fixbdr}] = deal( 1 );       % Set Dirchlet BCs for right boundary.
 fea.phys.el.bdr.coef{1,5} = bctype;

% Apply negative z-load to left edge.
 bccoef = num2cell( zeros(3,n_bdr) );
 [bccoef{3,forcebdr}] = deal(-opt.force);
 fea.phys.el.bdr.coef{1,end} = bccoef;


% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'fid', fid );


% Postprocessing.
 if ( opt.iplot>0 )
   DSCALE = 5000;

   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'sqrt(u^2+v^2+w^2)', 'linestyle', 'none' )
   title( 'Total displacement' )
   view([30 20])

   subplot(1,2,2)
   dp = zeros(size(fea.grid.p));
   for i=1:3
     dp(i,:) = DSCALE*evalexpr( fea.dvar{i}, fea.grid.p, fea );
   end
   fea_disp.grid   = fea.grid;
   fea_disp.grid.p = fea_disp.grid.p + dp;
   plotgrid( fea_disp )
   title(['Displacement plot (at ',num2str(DSCALE),' times scale)'])
   view([30 20])

 end


% Error check.
 disp_max_ref = 6.204e-6;
 xdisp = fea.sol.u(fea.eqn.dofm{1}(:));
 ydisp = fea.sol.u(fea.eqn.dofm{2}(:)+fea.eqn.ndof(1));
 zdisp = fea.sol.u(fea.eqn.dofm{3}(:)+sum(fea.eqn.ndof(1:2)));
 disp  = sqrt(xdisp.^2+ydisp.^2+zdisp.^2);
 disp_max = max(disp);

 svm_max_ref = 4.410e6;
 svm = evalexpr( fea.phys.el.eqn.vars{1,2}, fea.grid.p, fea );
 svm_max = max(svm);
```

```
 out.disp_max = disp_max;
 out.svm_max  = svm_max;
 out.err(1)   = abs(disp_max - disp_max_ref)/abs(disp_max_ref);
 out.err(2)   = abs(svm_max - svm_max_ref)/abs(svm_max_ref);
 out.pass     = all(out.err<opt.tol);


 if ( nargout==0 )
   clear fea out
 end


%
```

11.109 examples/ex_spanner

EX_SPANNER Gmsh grid import and stress calculation of a spanner.

[FEA, OUT] =

EX_SPANNER(VARARGIN) Example to import a Gmsh grid and calculate displacements on a fixed spanner. The load force may be distributed in the tangential load direction with the force fraction parameter FRAC.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {190e3}         Modulus of elasticity [N/mm^2]
nu          scalar {0.29}          Poissons ratio
force       scalar {1000}          Load force [N]
frac        scalar {0}             Fraction of stress against pulling direction
sfun        string {sflag1}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'E',        190e3;
             'nu',       0.29;
             'force',    1000;
             'frac',     0;
             'sfun',     'sflag1';
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});


% Define scaling factor m to mm.
 USE_METERS = true;
 s = double( ~USE_METERS + USE_METERS*1e-3 );


% Import (and scale) grid.
 fea.sdim = {'x','y','z'};
 fea.grid = impexp_gmsh( 'spannersh', 'import', [], [], opt.fid );
 fea.grid.p = fea.grid.p*s;


% Add linear elasticity physics mode and define material parameters.
 fea = addphys( fea, @linearelasticity );
 fea.phys.el.sfun            = { opt.sfun opt.sfun opt.sfun };
 fea.phys.el.eqn.coef{1,end} = { opt.nu };
 fea.phys.el.eqn.coef{2,end} = { opt.E/s^2 };


% Set all boundaries to no load per default.
 n_bdr  = max(fea.grid.b(3,:));
 bc_sel = cell(3,n_bdr);
 [bc_sel{:}] = deal(0);


% Fix all displacements on boundaries 11 and 14.
 i_fix = [11 14];
 [bc_sel{:,i_fix}] = deal(1);
 fea.phys.el.bdr.coef{5} = bc_sel;


% Apply force for x > 140 mm.
 i_force = [1 4];
 force = opt.force/(6*s*80*s);
 fea.phys.el.bdr.coef{7}{1,i_force(1)} = ['-',num2str((1-opt.frac)*force),'*(y>140*',num2str(s),')
 fea.phys.el.bdr.coef{7}{3,i_force(2)} = [num2str(opt.frac*force),'*(y>140*',num2str(s),')'];


% Parse and solve problem.
 fea = parsephys(fea);
 fea = parseprob(fea);
 fea.sol.u = solvestat( fea, 'fid', opt.fid, 'icub', 1+str2num(strrep(opt.sfun,'sflag','')) );
```

```
% Postprocessing.
 if( opt.iplot>0 )
   subplot(1,2,1)
   postplot( fea, 'surfexpr', ['sqrt(u^2+v^2+w^2)/',num2str(s)] )
   view([30 20])
   title('Total displacement (mm)')

   subplot(1,2,2)
   DSCALE = 5;
   dp = zeros(size(fea.grid.p));
   for i=1:3
     dp(i,:) = DSCALE*evalexpr( fea.dvar{i}, fea.grid.p, fea );
   end
   fea_disp.grid   = fea.grid;
   fea_disp.grid.p = fea_disp.grid.p + dp;
   plotgrid( fea_disp )
   title(['Displacement plot'])
   view([30 20])
 end


% Error checking.
 u = fea.sol.u(unique(fea.eqn.dofm{1}(:)))/s;
 v = fea.sol.u(unique(fea.eqn.ndof(1)+fea.eqn.dofm{2}(:)))/s;
 w = fea.sol.u(unique(sum(fea.eqn.ndof(1:2))+fea.eqn.dofm{3}(:)))/s;
 out.disp = sqrt( u.^2 + v.^2 + w.^2 );
 out.pass = nan;
 if( ~(got.frac || got.E || got.nu || got.force) )
   out.pass = abs( max(out.disp) - 2.5 )/2.5 < 0.1;
 end


 if( nargout==0 )
   clear fea out
 end
```

11.110 examples/ex_multiphase1

EX_MULTIPHASE1 Static bubble example.

[FEA, OUT] =

EX_MULTIPHASE1(VARARGIN) Sets up and solves a stationary static bubble example where the pressure jump should be equal to sigma/r. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {1e4}           Density
miu         scalar {1}             Molecular/dynamic viscosity
sigma       scalar {1}             Coefficient of surface tension
r           scalar {0.25}          Bubble radius
lbi         scalar 1/{0}           Exact or transformed surface tension source term
igrid       scalar 0/{1}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {1/30}          Max grid cell size
ischeme     scalar {3}             Time stepping scheme
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'rho',      1e4; ...
   'miu',      1; ...
   'sigma',    1; ...
   'r',        0.25; ...
   'lbi',      0; ...
   'igrid',    1; ...
   'hmax',     1/30; ...
   'ischeme'   3; ...
   'sf_u',     'sflag2'; ...
   'sf_p',     'sflag1'; ...
   'iplot',    1; ...
   'tol',      0.3; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' };   % Coordinate names.
 fea.grid = rectgrid(round(1/opt.hmax),round(2/opt.hmax),[0 1;0 1]);
 if ( opt.igrid==1 )
   fea.grid = quad2tri( fea.grid );
 end
 n_bdr = max(fea.grid.b(3,:)) + 1;   % Increment number of boundaries.
 fea.grid.b(3,1) = n_bdr;            % Set first boundary edge to boundary n_bdr.
```

```
% Problem definition.
 phi   = ['(sqrt((x-0.5)^2+(y-0.5)^2)-',num2str(opt.r),')'];   % Level set function.
 dh    = num2str(1*opt.hmax);   % Smoothing region width.
 dw    = ['(',phi,'/',dh,')'];
 smhs  = ['((0.5*(1+',dw,'+1/pi*sin(pi*',dw,')))*(',dw,'>-1)*(',dw,'<1)+(',dw,'>=1))'];
% Smooth heaviside function.
 smdel = ['0.5*(1+cos(pi*',dw,'))/',dh,'*(',dw,'>-1)*(',dw,'<1)'];
% Smooth delta function.
 sigma = num2str(opt.sigma);
 nx    = ['(2*x-1)/(2*sqrt((x-0.5)^2+(y-0.5)^2+eps))'];
 ny    = ['(2*y-1)/(2*sqrt((x-0.5)^2+(y-0.5)^2+eps))'];

% Surface tension force source term.
 if ( ~opt.lbi )   % Exact curvature.
   kappa = ['1/sqrt((x-0.5)^2+(y-0.5)^2+eps)'];
   fx1   = ['-',sigma,'*',kappa,'*',nx,'*',smdel];
   fy1   = ['-',sigma,'*',kappa,'*',ny,'*',smdel];
 else   % With Laplace-Beltrami transformation.
   fx1   = ['-',sigma,'*(1-(',nx,')^2)*',smdel];
   fx2   = ['-',sigma,'*(-(',nx,')*(',ny,'))*',smdel];
   fy1   = ['-',sigma,'*(-(',ny,')*(',nx,'))*',smdel];
   fy2   = ['-',sigma,'*(1-(',ny,')^2)*',smdel];
 end


% Add Navier-Stokes equations physics mode.
 fea = addphys(fea,@navierstokes);
 fea.phys.ns.eqn.coef{1,end} = { opt.rho };
 fea.phys.ns.eqn.coef{2,end} = { optiu };
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_p };
 fea.phys.ns.eqn.coef{3,end} = { fx1 };
 fea.phys.ns.eqn.coef{4,end} = { fy1 };
 fea.phys.ns.bdr.sel(n_bdr)  = 4;   % Set pressure to zero on last boundary segment.


% Parse problem.
 fea = parsephys(fea);
 if ( opt.lbi )
   fea.eqn.f.form{1} = [2 3];
   fea.eqn.f.form{2} = [2 3];
   fea.eqn.f.coef{1} = {fx1 fx2};
   fea.eqn.f.coef{2} = {fy1 fy2};
 end
 fea = parseprob(fea);


% Call to time-dependent solver.
 fea.sol.u = solvetime( fea, ...
                        'fid',     fid, ...
                        'tmax',    0.3, ...
```

```
                        'tstep',   0.1, ...
                        'icub',    3, ...
                        'ischeme', opt.ischeme );

% Postprocessing.
 if ( opt.iplot>0 )
   postplot(fea,'surfexpr','p')
   title('Pressure')
 end


% Error checking.
 ind_p    = [fea.eqn.ndof(1)+fea.eqn.ndof(2)+1:fea.eqn.ndof(1)+fea.eqn.ndof(2)+fea.eqn.ndof(3)];
 dp       = max(fea.sol.u(ind_p,end)) - min(fea.sol.u(ind_p,end));
 out.err  = abs(opt.sigma/opt.r - dp)/(opt.sigma/opt.r);
 out.pass = out.err<opt.tol;

 if ( nargout==0 )
   clear fea out
 end
```

11.111 examples/ex_multiphase2

EX_MULTIPHASE2 Multiphase flow rising bubble example.

[FEA, OUT] =

EX_MULTIPHASE2(VARARGIN) A benchmark multiphase flow rising bubble example. Reference solution is given in Hysing et. al Quantitative benchmark computations of two-dimensional bubble dynamics. Int. J. Numer. Meth. Fluids, 60: 1259–1288,

1. doi: 10.1002/fld.1934. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho1        scalar {1e2}           Density of the bubble
rho2        scalar {1e3}           Density of the surrounding fluid
miu1        scalar {1}             Viscosity of the bubble
miu2        scalar {1e1}           Viscosity of the surrounding fluid
gy          scalar {0.98}          Gravitational constant
sigma       scalar {24.5}          Coefficient of surface tension
igrid       scalar 0/{1}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {1/20}          Max grid cell size
dt          scalar {0.1}           Time step size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
```

```
sf_c        string {sflag1}        Shape function for the level set function
iplot       scalar 0/{1}           Plot solution and error (=1)

Output      Value/(Size)           Description
--------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'rho1',     1e2; ...
   'rho2',     1e3; ...
   'miu1',     1; ...
   'miu2',     1e1; ...
   'gy',       0.98; ...
   'sigma',    24.5; ...
   'igrid',    0; ...
   'hmax',     1/20; ...
   'dt',       0.1; ...
   'sf_u',     'sflag2'; ...
   'sf_p',     'sflag1'; ...
   'sf_c',     'sflag1'; ...
   'iplot',    1; ...
   'itest',    0; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' };   % Coordinate names.
 fea.grid = rectgrid(round(1/opt.hmax),round(2/opt.hmax),[0 1;0 2]);
 if ( opt.igrid==1 )
   fea.grid = quad2tri( fea.grid );
 end
 n_bdr = max(fea.grid.b(3,:)) + 1;   % Increment number of boundaries.
 fea.grid.b(3,1) = n_bdr;            % Set first boundary edge to boundary n_bdr.


% Problem definition.
 dh    = num2str(1.5*opt.hmax);      % Smoothing region width.
 dw    = ['(c/',dh,'/(sqrt(cx^2+cy^2+eps)))'];
 nx    = ['(cx/(sqrt(cx^2+cy^2+eps)))'];
 ny    = ['(cy/(sqrt(cx^2+cy^2+eps)))'];
```

```
 smhs  = ['((0.5*(1+',dw,'+1/pi*sin(pi*',dw,')))*(',dw,'>-1)*(',dw,'<1)+(',dw,'>=1))'];
% Smooth heaviside function.
 smdel = ['0.5*(1+cos(pi*',dw,'))/',dh,'*(',dw,'>-1)*(',dw,'<1)'];
% Smooth delta function.
 rho   = [num2str(opt.rho1),'+',num2str(opt.rho2-opt.rho1),'*',smhs];
 miu   = [num2str(optiu1),'+',num2str(optiu2-optiu1),'*',smhs];
 sigma = num2str(opt.sigma);

% Add Navier-Stokes equations physics mode.
 fea   = addphys(fea,@navierstokes);
 fea.phys.ns.eqn.coef{1,end} = { rho };
 fea.phys.ns.eqn.coef{2,end} = { miu };
 fea.phys.ns.bdr.sel(n_bdr)  = 4;   % Set pressure to zero on last boundary segment.
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_p };

% Source terms for gravity and surface tension effects.
 fg    = ['-(',rho,')*',num2str(opt.gy)];
 fx1   = ['-',sigma,'*(1-(',nx,')^2)*',smdel];
 fx2   = ['-',sigma,'*(-(',nx,')*(',ny,'))*',smdel];
 fy1   = ['-',sigma,'*(-(',ny,')*(',nx,'))*',smdel];
 fy2   = ['-',sigma,'*(1-(',ny,')^2)*',smdel];

% Add convection and diffusion physics mode for the level set equation.
 fea = addphys(fea,@convectiondiffusion);
 fea.phys.cd.sfun          = { opt.sf_c };
 fea.phys.cd.eqn.coef{2,4} = { 0.001 };    % Add stabilizing diffusion.
 fea.phys.cd.eqn.coef{3,4} = { 'u' };      % Convection velocity in the x-direction.
 fea.phys.cd.eqn.coef{4,4} = { 'v' };      % Convection velocity in the y-direction.


% Parse physics modes.
 fea = parsephys(fea);

% Correct source terms.
 fea.eqn.f.form{1} = [     2   3];
 fea.eqn.f.form{2} = [ 1   2   3];
 fea.eqn.f.coef{1} = {   fx1 fx2};
 fea.eqn.f.coef{2} = {fg fy1 fy2};

% Implement slip boundary conditions on vertical walls.
 fea.bdr.d{2}{2} = [];
 fea.bdr.d{2}{4} = [];
 fea.bdr.n{2}{2} = 0;
 fea.bdr.n{2}{4} = 0;

% Parse problem.
 fea = parseprob(fea);


% Call to time-dependent solver.
 init = { '0', '0', '0', 'sqrt((x-0.5)^2+(y-0.5)^2)-0.25' };
 fea.sol.u = solvetime( fea, ...
```

```
                        'fid',     fid, ...
                        'tmax',    3.0*(~opt.itest), ...
                        'tstep',   opt.dt, ...
                        'maxnit',  24*(~opt.itest)+1, ...
                        'icub',    3, ...
                        'init',    init, ...
                        'ischeme', 3 );

% Postprocessing.
 if ( opt.iplot>0 )
   subplot(2,2,1)
   postplot(fea,'surfexpr','sqrt(u^2+v^2)')
   title('Velocity field')
   subplot(2,2,2)
   postplot(fea,'surfexpr','p','evalstyle','exaxct')
   title('Pressure')
   subplot(2,2,3)
   postplot(fea,'surfexpr','c')
   title('Level set field')
   subplot(2,2,4)
   postplot(fea,'isoexpr','c','isolev',[0 0],'arrowexpr',{'u','v'})
   plot([0 1 1 0 0],[0 0 2 2 0],'k')
   title('Interface')
 end


 out.err  = [];
 out.pass = [];
 if ( nargout==0 )
   clear fea out
 end
```

11.112 examples/ex_multiphase3

EX_MULTIPHASE3 Breaking dam multiphase flow example.

[FEA, OUT] =

EX_MULTIPHASE3(VARARGIN) Breaking dam multiphase flow benchmark problem as define by Martin J, Moyce W. Part IV. An Experimental study of the collapse of liquid columns on a rigid horizontal plane. Phil. Trans. R. Soc. Lond. A 1952; 244(882):312–324, 1952, DOI: 10.1098/rsta.1952.0006. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho1        scalar {998}           Density of the water
rho2        scalar {1.205}         Density of the surrounding air
miu1        scalar {0.001}         Viscosity of the water
```

```
miu2        scalar {1.983e-5}      Viscosity of the surrounding air
gy          scalar {9.82}          Gravitational constant
a           scalar {0.05715}       Width and height of the water column
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {a/12}          Max grid cell size
dt          scalar {}              Time step size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
sf_c        string {sflag1}        Shape function for the level set function
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                                .
Output      Value/(Size)           Description
--------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
cOptDef = { ...
  'rho1',     998; ...
  'rho2',     1.205; ...
  'miu1',     0.001; ...
  'miu2',     1.983e-5; ...
  'gy',       9.82; ...
  'a',        0.05715; ...
  'igrid',    0; ...
  'hmax',     0.05715/18; ...
  'dt',       {}; ...
  'sf_u',     'sflag2';  ...
  'sf_p',     'sflag1'; ...
  'sf_c',     'sflag1'; ...
  'iplot',    1; ...
  'itest',    0; ...
  'fid',      1 };
[got,opt] = parseopt(cOptDef,varargin{:});
fid       = opt.fid;
if( ~got.dt )
  TSTEP = 0.05/sqrt(opt.gy/opt.a);
else
  TSTEP = opt.dt;
end
TMAX= 3.0/sqrt(opt.gy/opt.a);
MAXNIT = 30;
if( opt.itest )
  TMAX = TSTEP;
  MAXNIT = 1;
end
```

```
% Geometry and grid generation.
 l = 6*opt.a;
 h = 1.5*opt.a;
 fea.grid = rectgrid(round(l/opt.hmax),round(h/opt.hmax),[0 l;0 h]);
 if( opt.igrid==1 )
   fea.grid = quad2tri( fea.grid );
 end
 fea.sdim = { 'x' 'y' };   % Coordinate names.
 n_bdr = max(fea.grid.b(3,:)) + 1;   % Increment number of boundaries.
 fea.grid.b(3,1) = n_bdr;


% Problem definition.
 dh    = num2str(1.5*opt.hmax);      % Smoothing region width.
 dw    = ['(c/',dh,'/(sqrt(cx^2+cy^2+eps)))'];
 smhs  = ['((0.5*(1+',dw,'+1/pi*sin(pi*',dw,')))*(',dw,'>-1)*(',dw,'<1)+(',dw,'>=1))'];
% Smooth heaviside function.
 rho   = [num2str(opt.rho1),'+',num2str(opt.rho2-opt.rho1),'*',smhs];
 miu   = [num2str(optiu1),'+',num2str(optiu2-optiu1),'*',smhs,'+2e2*',num2str(opt.hmax),'*sqrt(u^2

% Add Navier-Stokes equations physics mode.
 fea   = addphys(fea,@navierstokes);
 fea.phys.ns.eqn.coef{1,end} = { rho };
 fea.phys.ns.eqn.coef{2,end} = { miu };
 fea.phys.ns.eqn.coef{4,end} = { ['-(',rho,')*',num2str(opt.gy)] };
 fea.phys.ns.bdr.sel(n_bdr)  = 4;   % Set pressure to zero on last boundary segment.
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_p };

 fea = addphys(fea,@convectiondiffusion);
 fea.phys.cd.sfun            = { opt.sf_c };
 fea.phys.cd.eqn.coef{2,4}   = { 0.001 };   % Set diffusion coefficient.
 fea.phys.cd.eqn.coef{3,4}   = { 'u' };    % Convection velocity in x-direction.
 fea.phys.cd.eqn.coef{4,4}   = { 'v' };    % Convection velocity in y-direction.


% Parse physics modes.
 fea = parsephys(fea);

% Implement slip boundary conditions on left and bottom walls.
 fea.bdr.d{1}{1} = [];
 fea.bdr.d{1}{5} = [];
 fea.bdr.d{2}{4} = [];
 fea.bdr.n{1}{1} = 0;
 fea.bdr.n{1}{5} = 0;
 fea.bdr.n{2}{5} = 0;

% Parse problem.
 fea = parseprob(fea);


% Call to time-dependent solver.
```

```
 init = { '0', '0', '0', ['(x-',num2str(opt.a),')*((x-',num2str(opt.a),')>(y-',num2str(opt.a),'))+
 fea.sol.u = solvetime( fea, ...
                        'fid',     fid, ...
                        'tmax',    TMAX, ...
                        'tstep',   TSTEP, ...
                        'icub',    3, ...
                        'maxnit',  MAXNIT, ...
                        'init',    init, ...
                        'ischeme', 3 );


% Postprocessing.
 if ( opt.iplot>0 )
   figure('position',[0 0 1920 1080])
   subplot(2,2,1)
   postplot(fea,'surfexpr','sqrt(u^2+v^2)')
   title('Velocity field')
   subplot(2,2,2)
   postplot(fea,'surfexpr','p','evalstyle','exaxct')
   title('Pressure')
   subplot(2,2,3)
   postplot(fea,'surfexpr','c')
   title('Level set field')
   subplot(2,2,4)
   postplot(fea,'surfexpr',rho)
% postplot(fea,'isoexpr','c','isolev',[0 0],'arrowexpr',{'u','v'})
   plot([0 l l 0 0],[0 0 h h 0],'k')
   title('Interface')
 end


 out.err  = [];
 out.pass = [];
 if ( nargout==0 )
   clear fea out
 end
```

11.113 examples/ex_heat_exchanger1

EX_FREE_CONVECTION Example of free and forced convection around a heated cylinder.

[FEA, OUT] =

EX_HEAT_EXCHANGER1(VARARGIN) Sets up and solves an example of temperature transport in a fluid around a heated cylinder. The model involves both free and forced convection by using the Boussinesq approximation.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {0.0005}        Max grid cell size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
sf_T        string {sflag1}        Shape function for temperature
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'hmax',     0.0005;
             'sf_u',     'sflag2';
             'sf_p',     'sflag1';
             'sf_T',     'sflag1';
             'iplot',    1;
             'w',        0.0075;
             'h',        0.05;
             'yc',       0.02;
             'r',        0.003;
             'rho',      2.2e1;
             'miu',      2.8e-3;
             'cp',       3.1e3;
             'k',        0.55;
             'al',       0.26e-3;
             'g',        9.81;
             'vin',      40e-2;
             'Tc',       300;
             'Th',       330;
             'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;


% Geometry definition.
 fea.sdim         = { 'x' 'y' };
 fea.geom.objects = { gobj_rectangle( 0, opt.w, 0, opt.h, 'R1' ) ...
                      gobj_circle( [0 opt.yc], opt.r, 'C1' ) };
 fea = geom_apply_formula( fea, 'R1-C1' );


% Grid generation.
```

```
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );


% Boundary conditions.
 dtol = 1e-6;
 ib_l = findbdr( fea, ['x<',num2str(dtol)] );         % Right boundary numbers.
 ib_r = findbdr( fea, ['x>',num2str(opt.w-dtol)] );   % Left boundary number.
 ib_b = findbdr( fea, ['y<',num2str(dtol)] );         % Bottom boundary number.
 ib_t = findbdr( fea, ['y>',num2str(opt.h-dtol)] );   % Top boundary number.
 ib_c = findbdr( fea, ['sqrt(x.^2+(y-',num2str(opt.yc),').^2)<',num2str(opt.r+dtol)] );
% Cylinder boundary numbers.


% Problem definition.
 fea.expr = { 'rho'   opt.rho ;
              'miu'   optiu ;
              'cp'    opt.cp  ;
              'k'     opt.k   ;
              'alpha' opt.al  ;
              'g'     opt.g   ;
              'vin'   opt.vin ;
              'Tc'    opt.Tc  ;
              'Th'    opt.Th  };

 fea = addphys(fea,@navierstokes);     % Add Navier-Stokes equations physics mode.
 fea.phys.ns.eqn.coef{1,end} = { 'rho' };
 fea.phys.ns.eqn.coef{2,end} = { 'miu' };
 fea.phys.ns.eqn.coef{4,end} = { 'alpha*g*rho*(T-Tc)' };
 fea.phys.ns.bdr.sel(ib_t)   = 4;      % Set pressure to zero on top boundary segment.
 fea.phys.ns.bdr.sel(ib_b)   = 2;      % Set y-velocity to vin at bottom boundary.
 fea.phys.ns.bdr.coef{2,end}{2,ib_b} = 'vin';
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_p };

 fea = addphys(fea,@heattransfer);     % Add heat transfer physics mode.
 fea.phys.ht.sfun            = { opt.sf_T };
 fea.phys.ht.eqn.coef{1,end} = { 'rho' };
 fea.phys.ht.eqn.coef{2,end} = { 'cp' };
 fea.phys.ht.eqn.coef{3,end} = { 'k' };
 fea.phys.ht.eqn.coef{4,end} = { fea.phys.ns.dvar{1} };
 fea.phys.ht.eqn.coef{5,end} = { fea.phys.ns.dvar{2} };
 fea.phys.ht.bdr.sel         = 3*ones( 1, max(fea.grid.b(3,:)) );   % Default to symmetry BCs.
 fea.phys.ht.bdr.sel(ib_t)   = 2;   % Set top boundary to convective flux/outflow.
 fea.phys.ht.bdr.sel([ib_c ib_b])    = 1;   % Set temperature to bottom and cylinder boundaries.
 [fea.phys.ht.bdr.coef{1,end}{ib_b}] = deal('Tc');
 [fea.phys.ht.bdr.coef{1,end}{ib_c}] = deal('Th');


% Parse and solve problem.
 fea = parsephys(fea);
 [fea.bdr.d{1}{[ib_l ib_r]}] = deal(0);    % Manually apply slip boundary conditions.
 [fea.bdr.d{2}{[ib_l ib_r]}] = deal([]);
 [fea.bdr.n{2}{[ib_l ib_r]}] = deal(0);
```

```
 fea = parseprob(fea);
 fea.sol.u = solvestat( fea, 'fid', fid, 'maxnit', 50 );


% Postprocessing.
 if( opt.iplot>0 )
   figure
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'sqrt(u^2+v^2)', 'arrowexpr', {'u' 'v'} )
   hold on
   fea.grid.p(1,:) = -fea.grid.p(1,:);   % Mirror solution.
   postplot( fea, 'surfexpr', 'sqrt(u^2+v^2)', 'arrowexpr', {'u' 'v'} )
   title('Velocity field')

   subplot(1,2,2)
   postplot( fea, 'surfexpr', 'T' )
   hold on
   fea.grid.p(1,:) = -fea.grid.p(1,:);   % Mirror solution.
   postplot( fea, 'surfexpr', 'T' )
   title('Temperature')
 end

% Average temperature at outlet.
 out.T_av_out = intbdr( 'T*v', fea, ib_t )/intbdr( 'v', fea, ib_t );


 if( nargout==0 )
   clear fea out
 end
```

11.114 examples/ex_natural_convection

EX_NATURAL_CONVECTION 2D Example for natural convection of air in a square cavity.

[FEA, OUT] =

EX_NATURAL_CONVECTION(VARARGIN) Sets up and solves a natural convection benchmark problem. Reference solutions are for example reported in G. Davis "Natural convection of air in a square cavity a bench mark numerical solution", IJNMF vol. 3, 249-254 (1983).

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
Ra          scalar {1e3}           Rayleigh number
Pr          scalar {0.71}          Prandtl number
l           scalar {1}             Side length of cavity
```

```
igrid       scalar 0/{1}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.05}          Max grid cell size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
sf_T        string {sflag2}        Shape function for temperature
iphys       scalar 0/{1}           Use physics mode to define problem (=1)
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                                .
Output      Value/(Size)           Description
--------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'Ra',       1e3; ...
   'Pr',       0.71; ...
   'l',        1; ...
   'igrid',    0; ...
   'hmax',     0.05; ...
   'sf_u',     'sflag2'; ...
   'sf_p',     'sflag1'; ...
   'sf_T',     'sflag2'; ...
   'iphys',    1; ...
   'iplot',    1; ...
   'tol',      0.1; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;

% Reference values.
 Ra_ref      = [ 1e3   1e4    1e5     1e6     ];
 u_max_ref   = [ 3.649 16.178 34.73   64.63   ];
 y_max_ref   = [ 0.813  0.823  0.855   0.850  ];
 v_max_ref   = [ 3.697 19.617 68.59  219.36   ];
 x_max_ref   = [ 0.178  0.119 0.066    0.0379 ];
 Nu_mean_ref = [ 1.118  2.243 4.519    8.800  ];

% Model parameters.
 Ra    = opt.Ra;      % Rayleigh number.
 Pr    = opt.Pr;      % Prandtl number.
 l     = opt.l;       % Length of rectangular domain.
 sf_u  = opt.sf_u;    % FEM shape function type for velocity.
 sf_p  = opt.sf_p;    % FEM shape function type for pressure.
 sf_T  = opt.sf_T;    % FEM shape function type for temperature.
```

```
% Geometry definition.
 fea.sdim         = { 'x' 'y' };
 fea.geom.objects = { gobj_rectangle( 0, l, 0, l ) };


% Grid generation.
 if ( opt.igrid==0 )
   fea.grid = rectgrid( round(l/opt.hmax), round(l/opt.hmax), [0 l;0 l] );
 else
   fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );
 end


% Boundary conditions.
 n_bdr = max(fea.grid.b(3,:));    % Increment number of boundaries.
 dtol  = opt.hmax;
 ib_l  = findbdr( fea, ['x<',num2str(dtol)] );     % Right boundary number.
 ib_r  = findbdr( fea, ['x>',num2str(l-dtol)] );   % Left boundary number.
 ib_b  = findbdr( fea, ['y<',num2str(dtol)] );     % Bottom boundary number.
 ib_t  = findbdr( fea, ['y>',num2str(l-dtol)] );   % Top boundary number.

% Add pressure point constraint on point closest to origin.
 [~,ix] = min( fea.grid.p(1,:).^2 + fea.grid.p(2,:).^2 );
 fea.pnt.index = ix;
 fea.pnt.type  = 'constr';
 fea.pnt.dvar  = 'p';
 fea.pnt.expr  = 0';


% Problem definition.
 if ( opt.iphys==1 )

   fea = addphys(fea,@navierstokes);     % Add Navier-Stokes equations physics mode.
   fea.phys.ns.eqn.coef{2,end} = { Pr };
   fea.phys.ns.eqn.coef{4,end} = { [num2str(Ra*Pr),'*T'] };
   fea.phys.ns.sfun            = { sf_u sf_u sf_p };   % Set shape functions.

   fea = addphys(fea,@heattransfer);     % Add heat transfer physics mode.
   fea.phys.ht.sfun            = { sf_T };
   fea.phys.ht.eqn.coef{4,end} = { fea.phys.ns.dvar{1} };
   fea.phys.ht.eqn.coef{5,end} = { fea.phys.ns.dvar{2} };
   fea.phys.ht.bdr.sel([ib_l ib_r])  = 1;
   fea.phys.ht.bdr.coef{1,end}{ib_l} = 1;

   fea = parsephys(fea);                 % Check and parse physics modes.

 else

   fea.dvar  = { 'u'  'v'  'p'  'T'  };       % Dependent variable name.
   fea.sfun  = { sf_u sf_u sf_p sf_T };       % Shape function.
```

```
% Define equation system.
   cvelx = fea.dvar{1};   % Convection velocities.
   cvely = fea.dvar{2};
   fea.eqn.a.form = { [2 3 2 3;2 3 1 1]      [2;3]                    [1;2] []; ...
                      [3;2]                  [2 3 2 3;2 3 1 1]        [1;3] []; ...
                      [2;1]                  [3;1]                    []    []; ...
                      []                     []                       []    [2 3 2 3;2 3 1 1] };
   fea.eqn.a.coef = { {2*Pr Pr cvelx cvely}   Pr                      -1 []; ...
                      Pr                     {Pr 2*Pr cvelx cvely}    -1 []; ...
                      1                       1                       [] []; ...
                      []                     []                       [] {1 1 cvelx cvely} };
   fea.eqn.f.form = { 1 1 1 1 };
   fea.eqn.f.coef = { 0 [num2str(Ra*Pr),'*',fea.dvar{4}] 0 0 };

% Define boundary conditions.
   fea.bdr.d = cell(4,n_bdr);
   [fea.bdr.d{1:2,:}] = deal(0);

   fea.bdr.d{4,ib_l}  = 1;
   fea.bdr.d{4,ib_r}  = 0;

   fea.bdr.n = cell(4,n_bdr);
   [fea.bdr.n{4,[ib_t ib_b n_bdr]}] = deal(0);
   [fea.bdr.n{3,setdiff(1:n_bdr,n_bdr)}] = deal(0);
   [fea.bdr.n{1:2,n_bdr}] = deal(0);

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid);   % Call to stationary solver.


% Postprocessing.
 if ( opt.iplot>0 )
   figure
   subplot(1,2,1)
   postplot(fea,'surfexpr','sqrt(u^2+v^2)')
   title('Velocity field')
   subplot(1,2,2)
   postplot(fea,'surfexpr','T')
   title('Temperature')
 end


% Error checking.
 out.err  = nan;
 out.pass = nan;
 iref = find( Ra==Ra_ref );
 if ( ~isempty(iref) )
   n_evalution_points = 3*l/opt.hmax;
```

```
    x_eval = linspace( 0, l, n_evalution_points );
    x_mid  = l/2*ones( 1, n_evalution_points );

    u_eval = evalexpr( 'u', [x_mid; x_eval], fea );
    [u_max,ix] = max( u_eval );
    y_max = x_eval( ix );

    v_eval = evalexpr( 'v', [x_eval; x_mid], fea );
    [v_max,ix] = max( v_eval );
    x_max = x_eval( ix );

    Nu_mean = abs( intbdr( 'Tx', fea, 1, 2 ) + intbdr( 'Tx', fea, 2, 2 ) )/2;

    out.u_max   = u_max;
    out.y_max   = y_max;
    out.v_max   = v_max;
    out.x_max   = x_max;
    out.Nu_mean = Nu_mean;

    out.err = [ abs(u_max_ref(iref)-u_max)/u_max_ref(iref);
                abs(y_max_ref(iref)-y_max)/y_max_ref(iref);
                abs(v_max_ref(iref)-u_max)/v_max_ref(iref);
                abs(x_max_ref(iref)-x_max)/x_max_ref(iref);
                abs(Nu_mean_ref(iref)-Nu_mean)/Nu_mean_ref(iref) ];
    out.pass = all( out.err<opt.tol );
  end

  if ( nargout==0 )
    clear fea out
  end
```

11.115 examples/ex_darcy1

EX_DARCY1 Darcy's law for a porous packed bed reactor.

[FEA, OUT] =

EX_DARCY1(VARARGIN) Example using Darcy's law to study the cross sectional flow in the porous material of a packed bed reactor.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {2e-4}          Grid size
sfun        string {sflag2}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
```

```
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'hmax',     2e-4;
             'sfun',     'sflag2';
             'iplot',    1;
             'tol',      5e-3;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Model constants.
 p0  = 1e5;       % Reference pressure.
 kap = 1.2e-13;   % Permeability.
 eta = 1.6e-5;    % Viscosity.
 M   = 16e-3;     % Molar mass.
 R   = 8.32;      % Ideal gas constant.
 T   = 344;       % Temperature.


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' };
 fea.geom.objects = { gobj_polygon([0     2.3e-3 2.3e-3 2.3e-3 2.3e-3 0    0    0
0;
                                    -6e-3 -6e-3  -5e-3  -4e-3   6e-3   6e-3 5e-3 3.5e-3 -6e-3]') };
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', opt.fid );


% Problem definition.
 fea = addphys( fea, @darcyslaw );   % Add Darcy's law physics mode.

% Modify equation kappa -> p*kappa (introduce nonlinearity in equation, not coefficient).
 fea.phys.dl.eqn.seqn = strrep(fea.phys.dl.eqn.seqn,'kap_dl','p*kap_dl');
 fea.phys.dl.eqn.coef{2,end} = { kap*M/R/T };
 fea.phys.dl.eqn.coef{3,end} = { eta };
 fea.phys.dl.sfun            = { opt.sfun };

% Set pressure 1.7 times higher on the inflow.
 i_inflow  = 7;
 i_outflow = 3;
 fea.phys.dl.bdr.sel(i_inflow)  = 1;
```

```
 fea.phys.dl.bdr.sel(i_outflow) = 1;
 fea.phys.dl.bdr.coef{1,end}{1,i_inflow}  = p0*1.7;
 fea.phys.dl.bdr.coef{1,end}{1,i_outflow} = p0;


% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );             % Check and parse problem struct.
 fea.sol.u = solvestat( fea, 'init', {p0}, 'fid', opt.fid );   % Call to stationary solver.


% Postprocessing.
 s_velmag = ['sqrt(((',num2str(-kap/eta),')^2)*(px^2+py^2))'];
 if( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', ['min(0.1,',s_velmag,')'], ...
                  'arrowexpr', {[num2str(-kap/eta),'*px'] [num2str(-kap/eta),'*py']} )
   title( 'Velocity field' )
 end


% Error checking.
 u_vel = intsubd( s_velmag, fea );
 p_in  = intbdr( 'p', fea, i_inflow );
 p_out = intbdr( 'p', fea, i_outflow );
 out.err = [ abs(u_vel-1.205e-6)/1.205e-6;
             abs(p_in-255)/255 ;
             abs(p_out-100)/100 ];
 out.pass = all( out.err < opt.tol );


 if ( nargout==0 )
   clear fea out
 end
```

11.116 examples/ex_navierstokes1

EX_NAVIERSTOKES1 2D Example for incompressible stationary flow in a channel.

[FEA, OUT] =

EX_NAVIERSTOKES1(VARARGIN) Sets up and solves stationary Poiseuille flow in a rectangular channel. The inflow profile is constant and the outflow should assume a parabolic profile (u(y)=U_max∗4/h^2∗y∗(h-y)). Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
```

```
rho         scalar {1}              Density
miu         scalar {0.001}          Molecular/dynamic viscosity
umax        scalar {0.3}            Maximum magnitude of inlet velocity
h           scalar {0.5}            Channel height
l           scalar {2.5}            Channel length
igrid       scalar 1/{0}            Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.04}           Max grid cell size
sf_u        string {sflag2}         Shape function for velocity
sf_p        string {sflag1}         Shape function for pressure
iphys       scalar 0/{1}            Use physics mode to define problem (=1)
solver      string <tt>openfoam</tt>/{'} Use OpenFOAM or default solver
iplot       scalar 0/{1}            Plot solution and error (=1)
                                                                                .
Output      Value/(Size)            Description
--------------------------------------------------------------------------------
fea         struct                  Problem definition struct
out         struct                  Output struct
```

Code listing

```
 cOptDef = { ...
   'rho',      1;
   'miu',      1e-3;
   'umax',     0.3;
   'h',        0.5;
   'l',        2.5;
   'igrid',    1;
   'hmax',     0.04;
   'sf_u',     'sflag2';
   'sf_p',     'sflag1';
   'iphys',    1;
   'solver',   '';
   'iplot',    1;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Model parameters.
 rho       = opt.rho;     % Density.
 miu       = optiu;     % Molecular/dynamic viscosity.
 umax      = opt.umax;    % Maximum magnitude of inlet velocity.
% Geometry and grid parameters.
 h         = opt.h;       % Height of rectangular domain.
 l         = opt.l;       % Length of rectangular domain.
% Discretization parameters.
 sf_u      = opt.sf_u;    % FEM shape function type for velocity.
```

```
 sf_p      = opt.sf_p;    % FEM shape function type for pressure.


% Geometry definition.
 gobj = gobj_rectangle( 0, l, 0, h );
 fea.geom.objects = { gobj };
 fea.sdim = { 'x' 'y' };   % Coordinate names.


% Grid generation.
 if ( opt.igrid==1 )
   fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 else
   fea.grid = rectgrid(round(l/opt.hmax),round(h/opt.hmax),[0 l;0 h]);
   if( opt.igrid<0 )
     fea.grid = quad2tri( fea.grid );
   end
 end
 n_bdr = max(fea.grid.b(3,:));           % Number of boundaries.


% Boundary conditions.
 dtol      = opt.hmax;
 i_inflow  = findbdr( fea, ['x<',num2str(dtol)] );     % Inflow boundary number.
 i_outflow = findbdr( fea, ['x>',num2str(l-dtol)] );   % Outflow boundary number.
 s_inflow  = ['2/3*',num2str(umax)];                                          % Definition of in
 s_refsol  = ['4*',num2str(umax),'*(y*(',num2str(h),'-y))/',num2str(h),'^2'];   % Definition of ve


% Problem definition.
 if ( opt.iphys==1 )

   fea = addphys(fea,@navierstokes);     % Add Navier-Stokes equations physics mode.
   fea.phys.ns.eqn.coef{1,end} = { rho };
   fea.phys.ns.eqn.coef{2,end} = { miu };
   fea.phys.ns.sfun            = { sf_u sf_u sf_p };          % Set shape functions.

   fea.phys.ns.bdr.sel(i_inflow)  = 2;
   fea.phys.ns.bdr.sel(i_outflow) = 4;
   fea.phys.ns.bdr.coef{2,end}{1,i_inflow} = s_inflow;         % Set inflow profile.
   fea = parsephys(fea);                 % Check and parse physics modes.

 else

   fea.dvar  = { 'u'  'v'  'p'  };       % Dependent variable name.
   fea.sfun  = { sf_u sf_u sf_p };       % Shape function.

% Define equation system.
   cvelx = [num2str(rho),'*',fea.dvar{1}];   % Convection velocity in x-direction.
   cvely = [num2str(rho),'*',fea.dvar{2}];   % Convection velocity in y-direction.
   fea.eqn.a.form = { [2 3 2 3;2 3 1 1]       [2;3]                    [1;2];
                      [3;2]                    [2 3 2 3;2 3 1 1]       [1;3];
```

```
                        [2;1]                    [3;1]                    []   };
   fea.eqn.a.coef = { {2*miu miu cvelx cvely}  miu                      -1;
                       miu                    {miu 2*miu cvelx cvely} -1;
                       1                       1                       [] };
   fea.eqn.f.form = { 1 1 1 };
   fea.eqn.f.coef = { 0 0 0 };


% Define boundary conditions.
   fea.bdr.d = cell(3,n_bdr);
  [fea.bdr.d{1:2,:}]         = deal( 0);

   fea.bdr.d{1,i_inflow}     = s_inflow;

  [fea.bdr.d{:,i_outflow  }] = deal([]);
% fea.bdr.d{end,i_outflow}  = 0;   % Set pressure to zero on outflow boundary.

   fea.bdr.n = cell(3,n_bdr);
 end


% Parse and solve problem.
 fea = parseprob(fea);             % Check and parse problem struct.
 if( opt.iphys==1 && strcmp(opt.solver,'openfoam') )
   fea = openfoam( fea );
 else
   jac.form  = {[1;1] [1;1] [];[1;1] [1;1] []; [] [] []};
   jac.coef  = {[num2str(rho),'*ux'] [num2str(rho),'*uy'] []; [num2str(rho),'*vx'] [num2str(rho),'
   fea.sol.u = solvestat( fea, 'fid', fid, 'nsolve', 2, 'jac', jac );   % Call to stationary solve
 end


% Postprocessing.
 s_velm = 'sqrt(u^2+v^2)';
 s_err  = ['abs(sqrt((',s_refsol,')^2)-(',s_velm,'))'];
 s_len  = ['(x>',num2str(3/4*l),')'];
 if ( opt.iplot>0 )
   figure
   subplot(3,1,1)
   postplot(fea,'surfexpr',s_velm,'evaltype','exact')
   title('Velocity field')
   subplot(3,1,2)
   postplot(fea,'surfexpr','p','evaltype','exact')
   title('Pressure')
   subplot(3,1,3)
   postplot(fea,'surfexpr',[s_err,'*',s_len],'evaltype','exact')
   title('Error')
 end


% Error checking.
 if ( size(fea.grid.c,1)==4 )
```

```
  xi = [0;0];
else
  xi = [1/3;1/3;1/3];
end
c_ind = find(evalexpr0(s_len,xi,1,1:size(fea.grid.c,2),[],fea))';
err = evalexpr0(s_err,xi,1,c_ind,[],fea);
ref = evalexpr0(['sqrt((',s_refsol,')^2)'],xi,1,c_ind,[],fea);
err = sqrt(sum(err.^2)/sum(ref.^2));


if( ~isempty(fid) )
  fprintf(fid,'\nL2 Error: %f\n',err)
  fprintf(fid,'\n\n')
end


out.err  = err;
out.pass = err<0.05;
if ( nargout==0 )
  clear fea out
end
```

11.117 examples/ex_navierstokes2

EX_NAVIERSTOKES2 2D Example for incompressible flow in a square cavity.

[FEA, OUT] =
EX_NAVIERSTOKES2(VARARGIN) Sets up and solves stationary

incompressible flow in a square cavity. References

[1] Botella O, Peyret R. Benchmark spectral results on the lid- driven cavity flow. Computers and Fluids 27(4):421–433, 1998.

[2] Erturk E, Corke TC, Gökcöl C. Numerical solutions of 2-D steady incompressible driven cavity flow at high Reynolds numbers. Int- ernational Journal for Numerical Methods in Fluids 37(6):633–655, 2005.

[3] Nishida H, Satofuka N. Higher-order solutions of square driven cavity flow using a variable-order multi-grid method. International Journal for Numerical Methods in Engineering 34(2):637–653, 1992.

[4] Schreiber R, Keller HB. Driven cavity flows by efficient numerical techniques. Journal of Computational Physics 49(2):310–333, 1983.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
re          scalar {1000}          Reynolds number
igrid       scalar {0}/1/2         Cell type (0=quadrilaterals, 1=triangles,
                                   2=triangles converted from quadrilaterals)
hmax        scalar {0.035}         Max grid cell size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iphys       scalar {1}/0           Use physics mode to define problem (=1)
solver      string <tt>openfoam</tt>/{'} Use OpenFOAM or default solver
iplot       scalar {1}/0           Plot solution and error (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   're',       1000;
   'igrid',    0;
   'hmax',     0.035;
   'sf_u',     'sflag2';
   'sf_p',     'sflag1';
   'iphys',    1;
   'solver',   '';
   'iplot',    1;
   'tol',      0.15;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Model parameters.
 rho       = 1;           % Density.
 umax      = 1;           % Maximum magnitude of inlet velocity.
 l         = 1;
 miu       = umax*l/opt.re;   % Molecular/dynamic viscosity.
% Grid parameters.
 hmax      = opt.hmax;    % Max allowable global element size.
```

```
 hmaxr     = 2*hmax;      % Max allowable element size on rectangle.
% Discretization parameters.
 sf_u      = opt.sf_u;    % FEM shape function type for velocity.
 sf_p      = opt.sf_p;    % FEM shape function type for pressure.


% Geometry definition.
 gobj = gobj_rectangle( 0, l, 0, l );
 fea.geom.objects = { gobj };
 fea.sdim = { 'x' 'y' };   % Coordinate names.


% Grid generation.
 switch opt.igrid
   case  0
     fea.grid = rectgrid(round(l/opt.hmax),round(l/opt.hmax),[0 l;0 l]);
   case  1
     fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
   case  2
     fea.grid = rectgrid(round(l/opt.hmax),round(l/opt.hmax),[0 l;0 l]);
     fea.grid = quad2tri(fea.grid,1);
 end


% Boundary conditions.
 n_bdr    = max(fea.grid.b(3,:));                    % Number of boundaries.
 dtol     = opt.hmax;
 i_inflow = findbdr( fea, ['y>',num2str(l-dtol)] );   % Inflow (top) boundary.


% Add pressure point constraint on point closest to origin.
 [~,ix] = min( fea.grid.p(1,:).^2 + fea.grid.p(2,:).^2 );
 fea.pnt.index = ix;
 fea.pnt.type  = 'constr';
 fea.pnt.dvar  = 'p';
 fea.pnt.expr  = 0';


% Problem definition.
 if ( opt.iphys==1 )

   fea = addphys(fea,@navierstokes);     % Add Navier-Stokes equations physics mode.
   fea.phys.ns.eqn.coef{1,end} = { rho };
   fea.phys.ns.eqn.coef{2,end} = { miu };
   fea.phys.ns.sfun            = { sf_u sf_u sf_p };          % Set shape functions.

   fea.phys.ns.bdr.sel(i_inflow) = 2;
   fea.phys.ns.bdr.coef{2,end}{1,i_inflow} = umax;            % Set inflow profile.
   fea = parsephys(fea);                % Check and parse physics modes.

 else
```

```
    fea.dvar  = { 'u'  'v'  'p'  };       % Dependent variable name.
    fea.sfun  = { sf_u sf_u sf_p };       % Shape function.

% Define equation system.
    cvelx = [num2str(rho),'*',fea.dvar{1}];   % Convection velocity in x-direction.
    cvely = [num2str(rho),'*',fea.dvar{2}];   % Convection velocity in y-direction.
    fea.eqn.a.form = { [2 3 2 3;2 3 1 1]       [2;3]                    [1;2];
                       [3;2]                   [2 3 2 3;2 3 1 1]       [1;3];
                       [2;1]                   [3;1]                    []   };
    fea.eqn.a.coef = { {2*miu miu cvelx cvely}  miu                     -1;
                        miu                    {miu 2*miu cvelx cvely} -1;
                        1                       1                       [] };
    fea.eqn.f.form = { 1 1 1 };
    fea.eqn.f.coef = { 0 0 0 };

% Define boundary conditions.
    fea.bdr.d = cell(3,n_bdr);
   [fea.bdr.d{1:2,:}]         = deal( 0);
    fea.bdr.d{1,i_inflow}     = umax;
    fea.bdr.n = cell(3,n_bdr);

 end


% Parse and solve problem.
 fea = parseprob(fea);             % Check and parse problem struct.
 if( opt.iphys==1 && strcmp(opt.solver,'openfoam') )
   fea = openfoam( fea );
 else
   fea.sol.u = solvestat(fea,'fid',fid);   % Call to stationary solver.
 end


% Postprocessing.
 s_velm = 'sqrt(u^2+v^2)';
 if ( opt.iplot>0 )
   figure
   subplot(3,1,1)
   postplot(fea,'surfexpr',s_velm,'evaltype','exact')
   title('Velocity field')
   subplot(3,1,2)
   postplot(fea,'surfexpr','p','evaltype','exact')
   title('Pressure')
   subplot(3,1,3)
   postplot(fea,'surfexpr','vx-uy','evaltype','exact','isoexpr',s_velm,'isolev',30)
   title('Vorticity')
 end


% Error checking.
 vort     = evalexpr('vx-uy',[0.53;0.564],fea);
 out.aerr = -2.068-vort;
```

```
  out.err  = abs(out.aerr)/2.068;
  out.pass = (abs(out.aerr)/2.068)<opt.tol;
  if ( nargout==0 )
    clear fea out
  end
```

11.118 examples/ex_navierstokes3

EX_NAVIERSTOKES3 2D Example for incompressible stationary flow around a cylinder.

[FEA, OUT] =EX_NAVIERSTOKES3(VARARGIN) Stationary flow around a cylinder. References

[1] John V, Matthies G. Higher-order finite element discretizations in a benchmark problem for incompressible flows. International Journal for Numerical Methods in Fluids 2001.

[2] Nabh G. On higher order methods for the stationary incompressible Navier-Stokes equations. PhD Thesis, Universitaet Heidelberg, 1998.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {1}             Density
miu         scalar {0.001}         Molecular/dynamic viscosity
umax        scalar {0.3}           Maximum magnitude of inlet velocity
igrid       scalar {2}             Grid type: >0 regular (igrid refinements)
                                               <0 unstruc. grid (with hmax=|igrid|)
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sf_disc1}      Shape function for pressure
iphys       scalar 0/{1}           Use physics mode to define problem (=1)
solver      string <tt>openfoam</tt>/{'} Use OpenFOAM or default solver
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
             'rho',      1;
             'miu',      0.001;
             'umax',     0.3;
             'igrid',    2;
             'sf_u',     'sflag2';
             'sf_p',     'sf_disc1';
             'iphys',    1;
             'solver',   '';
             'iplot',    1;
             'tol',      [0.01 0.2 0.01];
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Model parameters.
 rho       = opt.rho;     % Density.
 miu       = optiu;     % Molecular/dynamic viscosity.
 umax      = opt.umax;    % Maximum magnitude of inlet velocity.
 umean     = 2/3*umax;    % Mean inlet velocity.
% Geometry and grid parameters.
 h         = 0.41;        % Height of rectangular domain.
 l         = 2.2;         % Length of rectangular domain.
 xc        = 0.2;         % x-coordinate of cylinder center.
 yc        = 0.2;         % y-coordinate of cylinder center.
 diam      = 0.1;         % Diameter of cylinder.
% Discretization parameters.
 sf_u      = opt.sf_u;    % FEM shape function type for velocity.
 sf_p      = opt.sf_p;    % FEM shape function type for pressure.


% Grid generation.
 fea.sdim = { 'x' 'y' };
 if( opt.igrid>=1 )
   fea.grid = cylbenchgrid( opt.igrid );
   fea.grid.s(:) = 1;
 else
   gobj1 = gobj_rectangle( 0, 2.2, 0, 0.41, 'R1' );
   gobj2 = gobj_circle( [0.2 0.2], 0.05, 'C1' );
   fea.geom.objects = { gobj1 gobj2 };
   fea = geom_apply_formula( fea, 'R1-C1' );
   fea.grid = gridgen( fea, 'hmax', abs(opt.igrid), 'fid', fid );
 end
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Boundary conditions.
 dtol      = sqrt(eps)*1e3;
 i_inflow  = findbdr( fea, ['x<=',num2str(dtol)] );     % Inflow boundary number.
 i_outflow = findbdr( fea, ['x>=',num2str(l-dtol)] );   % Outflow boundary number.
```

```
 if( opt.iphys==1 && strcmp(opt.solver,'openfoam') )
   s_inflow  = num2str(2/3*umax);
 else
   s_inflow  = ['4*',num2str(umax),'*(y*(',num2str(h),'-y))/',num2str(h),'^2'];   % Definition of
 end
 i_cyl     = findbdr( fea, ['sqrt((x-',num2str(xc),').^2+(y-',num2str(yc),').^2)<=(',num2str(diam/
% Cylinder boundary number.


% Problem definition.
 if ( opt.iphys==1 )

   fea = addphys(fea,@navierstokes);     % Add Navier-Stokes equations physics mode.
   fea.phys.ns.eqn.coef{1,end} = { rho };
   fea.phys.ns.eqn.coef{2,end} = { miu };
   fea.phys.ns.sfun            = { sf_u sf_u sf_p };     % Set shape functions.

   fea.phys.ns.bdr.sel(i_inflow)  = 2;
   fea.phys.ns.bdr.sel(i_outflow) = 4;
   fea.phys.ns.bdr.coef{2,end}{1,i_inflow} = s_inflow;   % Set inflow profile.
   fea = parsephys(fea);                 % Check and parse physics modes.

 else

   fea.dvar  = { 'u'  'v'  'p'  };       % Dependent variable name.
   fea.sfun  = { sf_u sf_u sf_p };       % Shape function.

% Define equation system.
   cvelx = [num2str(rho),'*',fea.dvar{1}];   % Convection velocity in x-direction.
   cvely = [num2str(rho),'*',fea.dvar{2}];   % Convection velocity in y-direction.
   fea.eqn.a.form = { [2 3 2 3;2 3 1 1]       [2;3]                    [1;2];
                      [3;2]                    [2 3 2 3;2 3 1 1]       [1;3];
                      [2;1]                    [3;1]                   []   };
   fea.eqn.a.coef = { {2*miu miu cvelx cvely}  miu                     -1;
                      miu                     {miu 2*miu cvelx cvely} -1;
                      1                        1                      [] };
   fea.eqn.f.form = { 1 1 1 };
   fea.eqn.f.coef = { 0 0 0 };


% Define boundary conditions.
   fea.bdr.d = cell(3,n_bdr);
   [fea.bdr.d{1:2,:}]         = deal( 0);

   fea.bdr.d{1,i_inflow}      = s_inflow;

   [fea.bdr.d{:,i_outflow  }] = deal([]);

   fea.bdr.n = cell(3,n_bdr);
 end
```

```
% Parse and solve problem.
 fea = parseprob(fea);             % Check and parse problem struct.
 if( opt.iphys==1 && strcmp(opt.solver,'openfoam') )
   fea = openfoam( fea );
 else
   jac.form  = {[1;1] [1;1] [];[1;1] [1;1] []; [] [] []};
   jac.coef  = {[num2str(rho),'*ux'] [num2str(rho),'*uy'] []; [num2str(rho),'*vx'] [num2str(rho),'
   fea.sol.u = solvestat( fea, 'fid', fid, 'nsolve', 2, 'jac', jac, 'nlrlx', '(1+(it>2))/2' );
% Call to stationary solver.
 end

% Postprocessing.
 s_velm = 'sqrt(u^2+v^2)';
 if ( opt.iplot>0 )
   figure
   subplot(2,1,1)
   postplot( fea, 'surfexpr', s_velm )
   title( 'Velocity field' )
   subplot(2,1,2)
   postplot( fea, 'surfexpr', 'p' )
   title( 'Pressure' )
 end


% Calculate benchmark quantities (line integration method).
 s_tfx = ['nx*p+',num2str(miu),'*(-2*nx*ux-ny*(uy+vx))'];
 s_tfy = ['ny*p+',num2str(miu),'*(-nx*(vx+uy)-2*ny*vy)'];
 s_cd  = ['2*(',s_tfx,')/(',num2str(rho),'*',num2str(umean),'^2*',num2str(diam),')'];
 s_cl  = ['2*(',s_tfy,')/(',num2str(rho),'*',num2str(umean),'^2*',num2str(diam),')'];
 i_cub = 10;
 c_d1  = intbdr(s_cd,fea,i_cyl,i_cub);
 c_l1  = intbdr(s_cl,fea,i_cyl,i_cub);
 dp    = evalexpr('p',[0.15 0.25;0.2 0.2],fea);


% Calculate benchmark quantities (volume integration method).
 bdrm   = fea.bdr.bdrm{1};
 ind_b  = [];
 ind_bm = [];
 for ii=i_cyl
   ind_b  = [ind_b  find(fea.grid.b(3,:)==ii)];
   ind_bm = [ind_bm find(bdrm(3,:)==ii)];
 end
 ind_c    = fea.grid.b(1,ind_b);
 ind_gdof = bdrm(4,ind_bm);

% Create field 'a' with values one on the cylinder and zero everywhere else.
 fea.dvar = [ fea.dvar, {'a'}       ];
 fea.sfun = [ fea.sfun, fea.sfun(1) ];
 fea      = parseprob(fea);
 n_dof    = max(fea.eqn.dofm{1}(:));
 u_a      = zeros(n_dof,1);
```

```
  u_a(ind_gdof) = 1;
  fea.sol.u= [fea.sol.u;u_a];
  fea.eqn  = struct;
  fea.bdr  = struct;
  fea      = parseprob(fea);

  s_tfx    = ['ax*p+',num2str(miu),'*(-2*ax*ux-ay*(uy+vx))-(u*ux+v*uy)*a'];
  s_tfy    = ['ay*p+',num2str(miu),'*(-ax*(vx+uy)-2*ay*vy)-(u*vx+v*vy)*a'];
  s_cd     = ['2*(',s_tfx,')/(',num2str(rho),'*',num2str(umean),'^2*',num2str(diam),')'];
  s_cl     = ['2*(',s_tfy,')/(',num2str(rho),'*',num2str(umean),'^2*',num2str(diam),')'];
  c_d2     = intsubd(s_cd,fea,[],[],3);
  c_l2     = intsubd(s_cl,fea,[],[],3);


  if( ~isempty(fid) )
    fprintf(fid,'\n\nBenchmark quantities:\n\n')

    fprintf(fid,'Drag coefficient,    cd = %6f (l), %6f (v) (Ref: 5.579535)\n',c_d1,c_d2)
    fprintf(fid,'Lift coefficient,    cl = %6f (l), %6f (v) (Ref: 0.010619)\n',c_l1,c_l2)
    fprintf(fid,'Pressure,            dp = %6f (Ref: 0.117520)\n',dp(1)-dp(2))
  end


% Error checking.
  out.cd   = [c_d1 c_d2];
  out.cl   = [c_l1 c_l2];
  out.dp   = dp(1)-dp(2);
  out.err  = [abs(out.cd-5.579535)/5.579535;
              abs(out.cl-0.010619)/0.010619;
              abs(dp(1)-dp(2)-0.117520)/0.117520 0];
  out.pass = (out.err(1,2)<opt.tol(1))&&(out.err(2,2)<opt.tol(2))&&(out.err(3,1)<opt.tol(3));
  if ( nargout==0 )
    clear fea out
  end


%
```

11.119 examples/ex_navierstokes4

EX_NAVIERSTOKES4 2D Example for incompressible flow over a backwards facing step.

[FEA, OUT] =

EX_NAVIERSTOKES4(VARARGIN) Stationary flow over a backwards

facing step. References

[1] P.M. Gresho and R.L. Sani, Incompressible Flow and the Finite Element Method, Volume 1 & 2, John Wiley & Sons, New York, 2000.

[2] A. Rose and B. Simpson: “Laminar, Constant-Temperature Flow Over a Backward Facing Step,” 1st NAFEMS Workbook of CFD Examples, Glasgow, UK, 2000.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
re          scalar {389}           Reynolds number
h           scalar {1}             Channel height
y           scalar {0.485}         Step height (fraction of channel height)
lc          scalar {7.92}          Channel length (fraction of channel height)
li          scalar {1.98}          Inlet length (fraction of channel height)
hmax        scalar {0.06}          Max grid cell size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iphys       scalar 0/{1}           Use physics mode to define problem (=1)
solver      string <tt>openfoam</tt>/{'} Use OpenFOAM or default solver
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   're',       389;
   'h',        1;
   'y',        0.0049/0.0101;
   'lc',       0.08/0.0101;
   'li',       0.02/0.0101;
   'hmax',     0.1;
   'sf_u',     'sflag2';
   'sf_p',     'sflag1';
   'iphys',    1;
   'solver',   '';
   'iplot',    1;
   'fid',      1 };
```

```
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid parameters.
 h         = opt.h;       % Height of channel.
 y         = opt.y;       % Height of step.
 lc        = opt.lc;      % Length of channel.
 li        = opt.li;      % Length of inlet.
% Model parameters.
 umax      = 1;    % Maximum magnitude of inlet velocity.
 rho       = 1;           % Density.
 miu       = umax*2/3*h/opt.re;    % Molecular/dynamic viscosity.
% Discretization parameters.
 sf_u      = opt.sf_u;    % FEM shape function type for velocity.
 sf_p      = opt.sf_p;    % FEM shape function type for pressure.


% Geometry definition.
 vert = [ -li*h      lc*h lc*h    0 0 -li*h;   ...
          (1-y)*h (1-y)*h -y*h -y*h 0     0];
 gobj = gobj_polygon( vert' );
 fea.geom.objects = { gobj };
 fea.sdim = { 'x' 'y' };    % Coordinate names.


% Grid generation.
 fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 n_bdr    = max(fea.grid.b(3,:));        % Number of boundaries.


% Boundary conditions.
 dtol      = opt.hmax;
 i_inflow  = findbdr( fea, ['x<',num2str(-li*h+dtol)] );   % Inflow boundary number.
 i_outflow = findbdr( fea, ['x>',num2str( lc*h-dtol)] );   % Outflow boundary number.
 if( opt.iphys==1 && strcmp(opt.solver,'openfoam') )
   s_inflow  = num2str(2/3*umax);
 else
   s_inflow  = ['4*',num2str(umax),'*(y*(',num2str((1-y)*h),'-y))/',num2str((1-y)*h),'^2'];
% Definition of inflow profile.
 end


% Problem definition.
 if ( opt.iphys==1 )

   fea = addphys(fea,@navierstokes);     % Add Navier-Stokes equations physics mode.
   fea.phys.ns.eqn.coef{1,end} = { rho };
   fea.phys.ns.eqn.coef{2,end} = { miu };
   fea.phys.ns.sfun            = { sf_u sf_u sf_p };     % Set shape functions.

   fea.phys.ns.bdr.sel(i_inflow)  = 2;
```

```
    fea.phys.ns.bdr.sel(i_outflow) = 4;
    fea.phys.ns.bdr.coef{2,end}{1,i_inflow} = s_inflow;   % Set inflow profile.
    fea = parsephys(fea);                % Check and parse physics modes.

  else

    fea.dvar  = { 'u'  'v'  'p'  };       % Dependent variable name.
    fea.sfun  = { sf_u sf_u sf_p };       % Shape function.

% Define equation system.
    cvelx = [num2str(rho),'*',fea.dvar{1}];   % Convection velocity in x-direction.
    cvely = [num2str(rho),'*',fea.dvar{2}];   % Convection velocity in y-direction.
    fea.eqn.a.form = { [2 3 2 3;2 3 1 1]       [2;3]                    [1;2];
                       [3;2]                    [2 3 2 3;2 3 1 1]       [1;3];
                       [2;1]                    [3;1]                    []   };
    fea.eqn.a.coef = { {2*miu miu cvelx cvely}  miu                     -1;
                        miu                    {miu 2*miu cvelx cvely} -1;
                        1                       1                       [] };
    fea.eqn.f.form = { 1 1 1 };
    fea.eqn.f.coef = { 0 0 0 };


% Define boundary conditions.
    fea.bdr.d = cell(3,n_bdr);
   [fea.bdr.d{1:2,:}]         = deal( 0);

    fea.bdr.d{1,i_inflow}     = s_inflow;

   [fea.bdr.d{:,i_outflow  }] = deal([]);
% fea.bdr.d{end,i_outflow}  = 0;   % Set pressure to zero on outflow boundary.

    fea.bdr.n = cell(3,n_bdr);
  end


% Parse and solve problem.
  fea = parseprob(fea);             % Check and parse problem struct.
  if( opt.iphys==1 && strcmp(opt.solver,'openfoam') )
    fea = openfoam( fea );
  else
    fea.sol.u = solvestat(fea,'fid',fid,'maxnit',50,'nlrlx',1,'tolchg',1e-3);   % Call to stationar
  end

% Postprocessing.
  s_velm = 'sqrt(u^2+v^2)';
  if ( opt.iplot>0 )
    figure
    subplot(3,1,1)
    postplot(fea,'surfexpr',s_velm,'evaltype','exact','isoexpr',s_velm)
    title('Velocity field')
    subplot(3,1,2)
    postplot(fea,'surfexpr','p','evaltype','exact')
```

```
    title('Pressure')
    subplot(3,1,3)
    h = postplot(fea,'surfexpr',['(u<-eps)*x/',num2str(y)]);
    title('Separation length')
  end


% Error checking.
  s_expr = ['(u<-eps)*x/',num2str(y)];
  [~,slen] = minmaxsubd( s_expr, fea );
  if( ~isempty(fid) )
    fprintf(fid,'\nRecirculation zone length: %3f (Ref: 7.93)\n\n',slen)
    fprintf(fid,'\n\n')
  end

  out.slen = [slen 7.93];
  out.err  = abs(diff(out.slen))/out.slen(end);
  out.pass = out.err<0.2;
  if ( nargout==0 )
    clear fea out
  end
```

11.120 examples/ex_navierstokes5

EX_NAVIERSTOKES5 2D Vortex flow with analytical solution.

[FEA, OUT] =

EX_NAVIERSTOKES5(VARARGIN) Sets up and solves time dependent flow of a decaying vortex for which an analytical solution is known. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
nu          scalar {0.01}          Kinematic viscosity
k           scalar {pi/2}          Vortex number
xdim        {[-1 3]}               Domain min/max x-coordinates
ydim        {[-1 3]}               Domain min/max y-coordinates
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.2}           Max grid cell size
dt          scalar {0.01}          Time step size
tmax        icalar {0.1}           Simluation duration
ischeme     scalar {3}             Time stepping scheme
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                              .
Output      Value/(Size)           Description
```

```
-------------------------------------------------------------------------------
fea         struct                  Problem definition struct
out         struct                  Output struct
```

Code listing

```
 cOptDef = { ...
   'nu',       0.01; ...
   'k',        pi/2; ...
   'xdim',     [-1 3]; ...
   'ydim',     [-1 3]; ...
   'igrid',    0; ...
   'hmax',     0.2; ...
   'dt',       0.01; ...
   'tmax',     0.1; ...
   'ischeme',  3; ...
   'sf_u',     'sflag2'; ...
   'sf_p',     'sflag1'; ...
   'iplot',    1; ...
   'tol',      1e-1; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Exact solutions.
 k = num2str(opt.k);
 u_ref = ['-cos(',k,'*x).*sin(',k,'*y)*exp(-',num2str(2*opt.nu*opt.k^2),'*t)'];
 v_ref = [' sin(',k,'*x).*cos(',k,'*y)*exp(-',num2str(2*opt.nu*opt.k^2),'*t)'];
 p_ref = ['-1/4*(cos(2*',k,'*x)+cos(2*',k,'*y))*exp(-4*',num2str(opt.nu*opt.k^2),'*t)'];


% Grid generation.
 fea.sdim = { 'x' 'y' };
 if ( opt.igrid==0 )
   fea.grid = rectgrid( round(abs(diff(opt.xdim))/opt.hmax), round(diff(abs(opt.ydim))/opt.hmax),
 else
   fea.geom.objects = { gobj_rectangle( opt.xdim(1), opt.xdim(2), opt.ydim(1), opt.ydim(2) ) };
   fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );
 end


% Problem definition.
 fea = addphys( fea, @navierstokes );
 fea.phys.ns.eqn.coef{1,end} = { 1 };
 fea.phys.ns.eqn.coef{2,end} = { opt.nu };
 fea.phys.ns.sfun = { opt.sf_u opt.sf_u opt.sf_p };
```

```
  fea = parsephys( fea );


% Parse and solve problem.
  fea = parseprob( fea );
  [fea.bdr.d{1}{:}]  = deal( u_ref );   % Exact boundary conditions.
  [fea.bdr.d{2}{:}]  = deal( v_ref );
  [fea.bdr.d{3}{:}]  = deal( p_ref );
  fea.bdr.n = cell(3,max(fea.grid.b(3,:)));
  [fea.sol.u,tlist] = solvetime( fea, 'fid', fid, 'init', { u_ref v_ref p_ref }, 'ischeme', opt.isc


% Postprocessing.
  if ( opt.iplot>0 )
    subplot(1,2,1)
    postplot( fea, 'surfexpr', 'sqrt(u^2+v^2)', 'arrowexpr', {'u', 'v'}, 'arrowcolor', 'k' )
    title('Velocity field')

    subplot(1,2,2)
    postplot( fea, 'surfexpr', 'p' )
    title('Pressure')
  end


% Error checking.
  x = fea.grid.p(1,:)';
  y = fea.grid.p(2,:)';
  for i_sol=1:numel(tlist)
    t = tlist(i_sol);

    u_i = evalexpr( 'u', fea.grid.p, fea, i_sol );
    v_i = evalexpr( 'v', fea.grid.p, fea, i_sol );
    p_i = evalexpr( 'p', fea.grid.p, fea, i_sol );

    u_r = eval( u_ref );
    v_r = eval( v_ref );
    p_r = eval( p_ref );

    errnm(i_sol,1) = norm( u_i - u_r )/norm( u_r );
    errnm(i_sol,2) = norm( v_i - v_r )/norm( v_r );
    errnm(i_sol,3) = norm( p_i - p_r )/norm( p_r );
  end
  out.err  = errnm;
  out.pass = all( errnm(:)<opt.tol );


  if ( nargout==0 )
    clear fea out
  end
```

11.121 examples/ex_navierstokes6

EX_NAVIERSTOKES6 2D incompressible time-dependent flow around a cylinder.

[FEA, OUT] =

EX_NAVIERSTOKES6(VARARGIN) Benchmark example for time-dependent flow around a cylinder.

References

[1] John V. Reference values for drag and lift of a two-dimensional time-dependent flow around a cylinder. International Journal for Numerical Methods in Fluids 2004; 44:777-788.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {1}             Density
miu         scalar {0.001}         Molecular/dynamic viscosity
igrid       scalar {2}             Grid type: >0 regular (igrid refinements)
                                               <0 unstruc. grid (with hmax=|igrid|)
dt          scalar {0.02}          Time step size
ischeme     scalar {3}             Time stepping scheme
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sf_disc1}      Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                               .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'rho',      1; ...
   'miu',      0.001; ...
   'igrid',    2; ...
   'dt',       0.02; ...
   'ischeme'   3; ...
   'sf_u',     'sflag2'; ...
```

```
    'sf_p',     'sf_disc1'; ...
    'iplot',    1; ...
    'itest',    0; ...
    'fid',      1 };
  [got,opt] = parseopt(cOptDef,varargin{:});
  fid       = opt.fid;


% Grid generation.
  fea.sdim = { 'x' 'y' };
  if( opt.igrid>=1 )
    fea.grid = cylbenchgrid( opt.igrid );
  else
    gobj1 = gobj_rectangle( 0, 2.2, 0, 0.41, 'R1' );
    gobj2 = gobj_circle( [0.2 0.2], 0.05, 'C1' );
    fea.geom.objects = { gobj1 gobj2 };
    fea = geom_apply_formula( fea, 'R1-C1' );
    fea.grid = gridgen( fea, 'hmax', abs(opt.igrid), 'fid', fid );
  end


% Boundary conditions.
  inflow_boundary  = findbdr( fea, ['x<=',num2str(sqrt(eps))] );   % Inflow boundary number.
  outflow_boudnary = findbdr( fea, ['x>=',num2str(2.1)] );         % Outflow boundary number.
  inflow_bc = '6*sin(pi*t/8)*(y*(0.41-y))/0.41^2';


% Problem definition.
  fea = addphys( fea, @navierstokes );
  fea.phys.ns.eqn.coef{1,end} = { opt.rho };
  fea.phys.ns.eqn.coef{2,end} = { optiu };
  fea.phys.ns.sfun = { opt.sf_u opt.sf_u opt.sf_p };
  fea.phys.ns.bdr.sel( inflow_boundary  ) = 2;
  fea.phys.ns.bdr.sel( outflow_boudnary ) = 3;
  fea.phys.ns.bdr.coef{2,end}{1,inflow_boundary} = inflow_bc;
  fea = parsephys( fea );


% Call solver
  [fea.sol.u,tlist] = solvetime( fea, 'fid', fid, 'tstep', opt.dt, 'tmax', 8*(~opt.itest), 'maxnit'


% Benchmark quantities.
  [ c_d, c_l, dp ] = calc_bench_quant( fea, 1 );
  [c_d_max, i] = max( c_d );
  t_c_d_max    = tlist( i );
  [c_l_max, i] = max( c_l );
  t_c_l_max    = tlist( i );
  [~, i]       = min(abs(tlist-8));
  dp_t8        = dp(i);
  out.c_d = c_d;
  out.c_l = c_l;
```

```
  out.dp  = dp;
  out.c_d_max = c_d_max;
  out.c_l_max = c_l_max;
  out.t_c_d_max = t_c_d_max;
  out.t_c_l_max = t_c_l_max;
  out.dp_t8 = dp_t8;

  if( ~isempty(fid) )
    fmtf = '%12.8f |';
    fmts = '%12s |';
    fmt  = ['|      ',repmat(fmtf,1,5),'\n'];
    fmts = ['|      ',repmat(fmts,1,5),'\n'];
    fmtl = ['|------',repmat('-------------+',1,5)];
    fmtl = [fmtl(1:end-1),'|\n'];
    fprintf( fid, '\n\n' );
    fprintf( fid, fmtl );
    fprintf( fid, fmts, 't(cd_max)', 'cd_max', 't(cl_max)', 'cl_max', 'dp(t=8)' );
    fprintf( fid, fmtl );
    fprintf( fid, fmt, t_c_d_max, c_d_max, t_c_l_max, c_l_max, dp_t8 );
    fprintf( fid, fmtl );
    ref_data = [ 3.93625 2.950923849 5.6925 0.47834818 -0.11162153 ];
    fmt  = ['| Ref. ',repmat(fmtf,1,5),'\n'];
    fprintf( fid, fmt, ref_data );
    fprintf( fid, fmtl );
    fprintf( fid, '\n\n' );
  end


% Postprocessing.
  if( opt.iplot>0 )
    figure
    fea = parseprob( fea );

    subplot(2,2,1)
    postplot( fea, 'surfexpr', 'sqrt(u^2+v^2)', 'arrowexpr', {'u' 'v'}, 'arrowcolor', 'k' )
    title('Velocity field at t=8')

    subplot(2,2,3)
    plot( tlist, c_d )
    title('drag coefficient')

    subplot(2,2,4)
    plot( tlist, c_l )
    title('lift coefficient')

    subplot(2,2,2)
    plot( tlist, dp )
    title('pressure difference')
  end


  if( nargout==0 )
```

```
    clear fea out
  end


 %
```

11.122 examples/ex_navierstokes7

EX_NAVIERSTOKES7 3D Example for incompressible stationary flow in a curved pipe.

[FEA, OUT] =

EX_NAVIERSTOKES7(VARARGIN) Sets up and solves stationary flow in a curved circular channel. The inflow profile is constant and the outflow should assume an offset parabolic profile. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {1}             Density
miu         scalar {0.001}         Molecular/dynamic viscosity
umax        scalar {0.3}           Maximum magnitude of inlet velocity
h           scalar {0.5}           Channel radius
l           scalar {2.5}           Channel length
ilev        scalar {1}             Grid refinement level
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                               .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'rho',      1; ...
   'miu',      1e-3; ...
   'umax',     0.3; ...
   'h',        0.5; ...
   'l',        0.5; ...
   'ilev',     1; ...
```

```
    'sf_u',     'sflag2'; ...
    'sf_p',     'sflag1'; ...
    'iplot',    1; ...
    'fid',      1 };
  [got,opt] = parseopt(cOptDef,varargin{:});
  fid       = opt.fid;


% Grid generation.
  fea.sdim = { 'x' 'y' 'z' };   % Coordinate names.
  fea.grid = gridrevolve( circgrid( 4*opt.ilev, 3*opt.ilev, opt.h/2 ), 15*opt.ilev, opt.l, 1/4 );


% Problem definition.
  fea = addphys( fea, @navierstokes );
  fea.phys.ns.eqn.coef{1,end} = { opt.rho };
  fea.phys.ns.eqn.coef{2,end} = { optiu };
  fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_u opt.sf_p };


% Boundary conditions.
  fea.phys.ns.bdr.sel(1)  = 2;
  fea.phys.ns.bdr.sel(18) = 3;
  fea.phys.ns.bdr.coef{2,end}{2,1} = -opt.umax;


% Parse and solve problem.
  fea       = parsephys(fea);
  fea       = parseprob(fea);
  jac.form  = { [1;1] [1;1] [1;1] []; [1;1] [1;1] [1;1] [];  [1;1] [1;1] [1;1] []; [] [] [] [] };
  jac.coef  = { 'rho_ns*ux' 'rho_ns*uy' 'rho_ns*uz' []; 'rho_ns*vx' 'rho_ns*vy' 'rho_ns*vz' []; 'rh
  fea.sol.u = solvestat( fea, 'fid', fid, 'nsolve', 2, 'jac', jac );


% Postprocessing.
  if( opt.iplot>0 )
    postplot( fea, 'surfexpr', 'sqrt(u^2+v^2+w^2)' )
    view( 130, 30 )
  end


% Error checking.
  out.flow_in  = pi*(opt.h/2)^2*opt.umax;
  out.flow_out = intbdr( 'sqrt(u^2+v^2+w^2)', fea, 18 );
  out.rerr = abs(out.flow_out-out.flow_in)/out.flow_in;
  out.pass = out.rerr < 0.12;


  if ( nargout==0 )
    clear fea out
  end
```

11.123 examples/ex_navierstokes8

EX_NAVIERSTOKES8 2D Example for axisymmetric incompressible stationary flow in a constricted circular pipe.

[FEA, OUT] =

EX_NAVIERSTOKES8(VARARGIN) Sets up and solves stationary axisymmetric Poiseuille flow in a constricted circular pipe. The inflow profile is constant and the outflow should assume a parabolic profile. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {1}             Density
miu         scalar {1}             Molecular/dynamic viscosity
uin         scalar {1}             Inflow velocity (constant/mean)
r           scalar {1}             Channel radius
l           scalar {3}             Channel length
hmax        scalar {0.1}           Max grid cell size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'rho',      1; ...
   'miu',      1; ...
   'uin',      1; ...
   'r',        1; ...
   'l',        3; ...
   'hmax',     0.1; ...
   'sf_u',     'sflag2'; ...
   'sf_p',     'sflag1'; ...
   'iplot',    1; ...
   'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;
```

```
% Geometry definition.
 r = opt.r;   % Pipe radius.
 l = opt.l;   % Pipe length.
 gobj1 = gobj_rectangle( 0, r,   0,     l*2/3, 'R1' );
 gobj2 = gobj_rectangle( 0, r/2, l*2/3, l,     'R2' );
 gobj3 = gobj_circle( [r l*2/3], r/2, 'C1' );
 fea.geom.objects = { gobj1 gobj2 gobj3 };
 fea = geom_apply_formula( fea, 'R1+R2-C1' );
 fea.sdim = { 'r' 'z' };   % Space coordinate names.


% Grid generation.
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );


% Boundary specifications.
 i_inflow   = 1;       % Inflow boundary number.
 i_outflow  = 5;       % Outflow boundary number.
 i_symmetry = [3 6];   % Symmetry boundary numbers.


% Problem definition.
 fea = addphys( fea, {@navierstokes 1} );   % Add Navier-Stokes equations physics mode.
 fea.phys.ns.eqn.coef{1,end} = { opt.rho };
 fea.phys.ns.eqn.coef{2,end} = { optiu };
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_p };


% Boundary conditions.
 fea.phys.ns.bdr.sel(i_inflow)   = 2;
 fea.phys.ns.bdr.sel(i_outflow)  = 3;
 fea.phys.ns.bdr.coef{2,end}{2,i_inflow} = opt.uin;
 fea.phys.ns.bdr.sel(i_symmetry) = 5;
 fea = parsephys(fea);  % Parse physics mode.


% Parse and solve problem.
 fea = parseprob(fea);   % Check and parse problem struct.
 jac.form  = {[1;1] [1;1] [];[1;1] [1;1] []; [] [] []};
 jac.coef  = {'r*rho_ns*ur' 'r*rho_ns*uz' []; 'r*rho_ns*wr' 'r*rho_ns*wz' []; [] [] []};
 fea.sol.u = solvestat( fea, 'fid', fid, 'nsolve', 2, 'jac', jac );


% Error checking.
 r = linspace( 0, opt.r/2, 20 );
 z = 0.9*opt.l*ones( 1, 20 );
 U = evalexpr( 'sqrt(u^2+w^2)', [r;z], fea )';
 u_fac = 4;   % Due to contraction to 1/2 radius.
 U_ref = 2*opt.uin*u_fac*( 1 - ( r/(opt.r/2) ).^2 );
 err = sqrt( sum((U-U_ref).^2)/sum(U_ref.^2) );
```

```
% Postprocessing.
 if( opt.iplot>0 )
   figure
   subplot(1,3,1)
   postplot( fea, 'surfexpr', 'sqrt(u^2+w^2)', 'arrowexpr', {'u' 'w'} )
   hold on
   plot( r, z, 'k--' )
   title( 'Velocity field' )

   subplot(1,3,2)
   postplot( fea, 'surfexpr', 'p', 'evaltype', 'exact', 'isoexpr', 'p' )
   title( 'Pressure' )

   subplot(1,3,3)
   plot( r, U,     'b-' )
   hold on
   plot( r, U_ref, 'r-' )
   title('Velcity profile at z=0.9*l')
   xlabel( 'Radius' )
   legend( 'Computed', 'Reference', 'location', 'south' )
 end


 out.err  = err;
 out.pass = err<0.05;
 if( nargout==0 )
   clear fea out
 end
```

11.124 examples/ex_navierstokes9

EX_NAVIERSTOKES9 2D Axisymmetric jet impingement with heat transfer.

[FEA, OUT] =

EX_NAVIERSTOKES9(VARARGIN) Sets up and solves stationary axisymmetric flow of a jet impacting in a thin constrained region with a coupled heat transfer mode. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
Re          scalar {100}           Jet Reynolds number
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.2}           Max grid cell size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution and error (=1)
```

```
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'Re',       100; ...
   'igrid',    0; ...
   'hmax',     0.2; ...
   'sf_u',     'sflag2'; ...
   'sf_p',     'sflag1'; ...
   'iplot',    1; ...
   'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;


% Geometry definition.
 r = 10;   % Radius.
 l = 3;    % Length.
 gobj = gobj_polygon( [0 0; r 0; r l; 1 l; 0 l] );
 fea.geom.objects = { gobj };
 fea.sdim = { 'r' 'z' };   % Space coordinate names.


% Grid generation.
 if ( opt.igrid==1 )
   fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );
 else
   fea.grid = rectgrid( round(r/opt.hmax), round(l/opt.hmax), [0 r;0 l] );
   if( opt.igrid<0 )
     fea.grid = quad2tri( fea.grid );
   end
   ic = fea.grid.b(1,:);
   ii = fea.grid.b(2,:);
   jj = mod(ii,size(fea.grid.c,1)) + 1;
   p1 = fea.grid.p( :, fea.grid.c(sub2ind(size(fea.grid.c),ii,ic)) );
   p2 = fea.grid.p( :, fea.grid.c(sub2ind(size(fea.grid.c),jj,ic)) );
   pm = [ p1 + p2 ]'/2;

   ix4 = find( (pm(:,1)<1) .* (abs(pm(:,2)-3)<eps) );
   fea.grid.b(3,ix4) = 4;

   ix5 = find( pm(:,1)<=eps );
```

```
    fea.grid.b(3,ix5) = 5;
  end


% Boundary specifications.
  i_plate    = 1;
  i_inflow   = 4;
  i_outflow  = 2;
  i_symmetry = 5;
  inflow_bc  = -opt.Re;


% Problem definition.

% Add Navier-Stokes equations physics mode.
  fea = addphys( fea, {@navierstokes 1} );
  fea.phys.ns.eqn.coef{1,end} = { 1 };
  fea.phys.ns.eqn.coef{2,end} = { 1 };
  fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_p };


  fea.phys.ns.bdr.sel(i_inflow)   = 2;
  fea.phys.ns.bdr.sel(i_outflow)  = 3;
  fea.phys.ns.bdr.coef{2,end}{2,i_inflow} = inflow_bc;
  fea.phys.ns.bdr.sel(i_symmetry) = 5;

% Add heat transfer physics mode.
  fea = addphys( fea, {@heattransfer 1} );
  fea.phys.ht.eqn.coef{4,end} = fea.phys.ns.dvar{1};
  fea.phys.ht.eqn.coef{5,end} = fea.phys.ns.dvar{2};
  fea.phys.ht.sfun            = { opt.sf_u };

  fea.phys.ht.bdr.sel(:)         = 3;
  fea.phys.ht.bdr.sel(i_inflow)  = 1;
  fea.phys.ht.bdr.sel(i_plate)   = 1;
  fea.phys.ht.bdr.sel(i_outflow) = 2;
  fea.phys.ht.bdr.coef{1,end}{1,i_inflow} = 1;

% Parse physics modes.
  fea = parsephys(fea);


% Parse and solve problem.
  fea       = parseprob( fea );   % Check and parse problem struct.
  jac.form  = {[1;1] [1;1] [] [];[1;1] [1;1] [] []; [] [] [] []; [] [] [] []};
  jac.coef  = {'r*rho_ns*ur' 'r*rho_ns*uz' [] []; 'r*rho_ns*wr' 'r*rho_ns*wz' [] []; [] [] [] [];[]
  fea.sol.u = solvestat( fea, 'fid', fid, 'nsolve', 2, 'jac', jac );


% Error checking.
  u = evalexpr( 'u', fea.grid.p, fea );
  w = evalexpr( 'w', fea.grid.p, fea );
```

```
 out.err(1) = (max(u) - 75.9)/75.9;
 out.err(2) = (min(u) +  9.8)/(-9.8);
 out.err(3) = (max(w) -  7.9)/7.9;
 out.err(4) = (min(w) +  106)/(-106);
 out.pass   = all( out.err<0.05 );


% Postprocessing.
 if( opt.iplot>0 )
   figure
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'sqrt(u^2+w^2)', ...
                  'arrowexpr', {'u' 'w'}, 'arrowspacing', [60 20], ...
                  'isoexpr', 'sqrt(u^2+w^2)' )
   title( 'Velocity field' )

   subplot(1,2,2)
   postplot( fea, 'surfexpr', 'T', 'isoexpr', 'T' )
   title( 'Temperature' )
 end


 if( nargout==0 )
   clear fea out
 end
```

11.125 examples/ex_navierstokes10

EX_NAVIERSTOKES10 3D Example for stationary flow in a pipe.

[FEA, OUT] =

EX_NAVIERSTOKES10(VARARGIN) Sets up and solves stationary and laminar 3D flow in a circular pipe. The inflow profile is constant and the outflow should assume an offset parabolic profile. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {1}             Density
miu         scalar {0.01}          Molecular/dynamic viscosity
uin         scalar {0.3}           Magnitude of inlet velocity
R           scalar {0.5}           Channel radius
sf_u        string {sf_hex_Q1nc}   Shape function for velocity
sf_p        string {sf_disc0}      Shape function for pressure
solver      string <tt>openfoam</tt>/{'} Use OpenFOAM or default solver
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                              .
```

```
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = {   ...
   'rho',      1;
   'miu',      1e-2;
   'uin',      0.3;
   'R',        0.5;
   'sf_u',     'sf_hex_Q1nc';
   'sf_p',     'sf_disc0';
   'tol',      0.1;
   'solver',   'openfoam';
   'iplot',    1;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x', 'y', 'z' };
 fea.geom.objects = { gobj_cylinder([0 0 0],opt.R,3,1) };
 fea.grid = cylgrid(4,4,20,opt.R,3,[0;0;0],1);


% Problem definition.
 fea = addphys( fea, @navierstokes );
 fea.phys.ns.eqn.coef{1,end} = { opt.rho };
 fea.phys.ns.eqn.coef{2,end} = { optiu };
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_u opt.sf_p };


% Boundary conditions.
 fea.phys.ns.bdr.sel(5) = 2;
 fea.phys.ns.bdr.sel(6) = 4;
 fea.phys.ns.bdr.coef{2,end}{1,5} = opt.uin;


% Parse and solve problem.
 fea = parsephys( fea );
 fea = parseprob( fea );
 if( strcmp(opt.solver,'openfoam') )
   fea = openfoam( fea );
 else
```

```
    fea.sol.u = solvestat( fea, 'fid', fid );
  end


% Postprocessing.
  if( opt.iplot>0 )
    postplot( fea, 'sliceexpr', 'sqrt(u^2+v^2+w^2)' )
  end


% Error checking.
  n = 15;
  y = linspace(0.05,0.95,n)' - 0.5;
  p = repmat([3 0 0]',1,n);
  p(2,:) = y;
  u = evalexpr( 'u', p, fea );
  u_ref = 2*opt.uin*(1-(y/opt.R).^2);
  out.err  = mean(abs(u-u_ref)./u_ref);
  out.pass = out.err < opt.tol;


  if ( nargout==0 )
    clear fea out
  end
```

11.126 examples/ex_navierstokes11

EX_NAVIERSTOKES11 3D Example flow in a cubcic cavity.

[FEA, OUT] =

EX_NAVIERSTOKES11(VARARGIN) Sets up and solves stationary and laminar 3D flow in a cubic cavity. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {1}             Density
miu         scalar {1}             Molecular/dynamic viscosity
uin         scalar {1}             Magnitude of inlet velocity
sf_u        string {sf_hex_Q1nc}   Shape function for velocity
sf_p        string {sf_disc0}      Shape function for pressure
solver      string <tt>openfoam</tt>/{'} Use OpenFOAM or default solver
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                          .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
```

```
out         struct                Output struct
```

Code listing

```
 cOptDef = {   ...
   'rho',      1;
   'miu',      1;
   'uin',      1;
   'sf_u',     'sf_hex_Q1nc';
   'sf_p',     'sf_disc0';
   'tol',      0.15;
   'solver',   'openfoam';
   'iplot',    1;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x', 'y', 'z' };
 fea.geom.objects = { gobj_block() };
 fea.grid = blockgrid(8);


% Problem definition.
 fea = addphys( fea, @navierstokes );
 fea.phys.ns.eqn.coef{1,end} = { opt.rho };
 fea.phys.ns.eqn.coef{2,end} = { optiu };
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_u opt.sf_p };


% Boundary conditions.
 fea.phys.ns.bdr.sel(6) = 2;
 fea.phys.ns.bdr.coef{2,end}{1,6} = opt.uin;


% Parse and solve problem.
 fea  = parsephys( fea );
 fea  = parseprob( fea );
 if( strcmp(opt.solver,'openfoam') )
   fea = openfoam( fea );
 else
% Add pressure point constraint on point closest to origin.
   [~,ix] = min( fea.grid.p(1,:).^2 + fea.grid.p(2,:).^2 + fea.grid.p(3,:).^2 );
   fea.pnt.index = ix;
   fea.pnt.type  = 'constr';
   fea.pnt.dvar  = 'p';
```

```
    fea.pnt.expr  = 0';

    fea.sol.u = solvestat( fea, 'fid', fid );
  end


% Postprocessing.
  if( opt.iplot>0 )
    postplot( fea, 'sliceexpr', 'sqrt(u^2+v^2+w^2)' )
  end


% Error checking.
  u = evalexpr( 'u', [0.5;0.5;0.5], fea );
  out.err  = [u-(-0.21)]/0.21;
  out.pass = out.err < opt.tol;


  if ( nargout==0 )
    clear fea out
  end
```

11.127 examples/ex_navierstokes12

EX_NAVIERSTOKES12 3D Example flow over a backwards facing step.

[FEA, OUT] =

EX_NAVIERSTOKES12(VARARGIN) Sets up and solves stationary and laminar 3D flow over a backwards facing step. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-----------------------------------------------------------------------------------
rho         scalar {1}             Density
miu         scalar {2/3/389}       Molecular/dynamic viscosity
uin         scalar {1}             Magnitude of inlet velocity
sf_u        string {sf_hex_Q1nc}   Shape function for velocity
sf_p        string {sf_disc0}      Shape function for pressure
solver      string <tt>openfoam</tt>/{'} Use OpenFOAM or default solver
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                                  .
Output      Value/(Size)           Description
-----------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = {   ...
   'rho',      1;
   'miu',      2/3/389;
   'uin',      1;
   'igrid',    1;
   'sf_u',     'sf_hex_Q1nc';
   'sf_p',     'sf_disc0';
   'tol',      0.2;
   'solver',   'openfoam';
   'iplot',    1;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry.
 fea.sdim = { 'x', 'y', 'z' };
 gobj1 = gobj_block( -1.9802, 7.9208, 0, 1, 0, 1, 'B1' );
 gobj2 = gobj_block( -1.9802, 0, 0, 1, 0, 0.4851, 'B2' );
 fea.geom.objects = { gobj1 gobj2 };
 fea = geom_apply_formula( fea, 'B1-B2' );


% Grid generation.
 if( opt.igrid==1 )
   n = 4;
   fea.grid = rectgrid(10*n,n,[-1.9802, 7.9208;0 1]);
   fea.grid = delcells( fea.grid, selcells(fea.grid,'(y<=0.5).*(x<=0)') );
   ix = find( abs(fea.grid.p(2,:)-0.5)<=sqrt(eps) );
   fea.grid.p(2,ix) = 0.4851;
   fea.grid = gridextrude( fea.grid, n, 1, -2 );
   fea.grid.p(2,:) = fea.grid.p(2,:) + 1;
   fea.grid = assign_bdr( fea.grid, fea.geom, [] );
   [~,ix] = findbdr( fea , 'z>=1-sqrt(eps)', 0 );
   fea.grid.b(3,ix) = 6;
 else
   fea.grid = gridgen( fea, 'hmax', 0.4, 'fid', [] );
   fea.grid = gridsmooth( tet2hex( fea.grid ), 5 );
 end


% Problem definition.
 fea = addphys( fea, @navierstokes );
 fea.phys.ns.eqn.coef{1,end} = { opt.rho };
 fea.phys.ns.eqn.coef{2,end} = { optiu };
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_u opt.sf_p };
```

```
% Boundary conditions.
 fea.phys.ns.bdr.sel(5) = 2;
 fea.phys.ns.bdr.sel(3) = 4;
 if( strcmp(opt.solver,'openfoam') )
   fea.phys.ns.bdr.coef{2,end}{1,5} = 3/2*opt.uin;
 else
   fea.phys.ns.bdr.coef{2,end}{1,5} = ['4*',num2str(opt.uin),'*(z-0.4851)*(1-z)/(1-0.4851)^2'];
 end


% Parse and solve problem.
 fea  = parsephys( fea );
 fea  = parseprob( fea );
 if( strcmp(opt.solver,'openfoam') )
   fea = openfoam( fea );
 else
   fea.sol.u = solvestat( fea, 'fid', fid );
 end


% Postprocessing.
 if( opt.iplot>0 )
   postplot( fea, 'sliceexpr', 'sqrt(u^2+v^2+w^2)' )
 end


% Error checking.
 [~,slen] = minmaxsubd( '(u<-eps)*x/0.4851*(z<0.4851)*(z>0)*(y<0.51)*(y>0.49)', fea );
 out.err  = abs( slen - 3 )/3;
 out.pass = out.err < opt.tol;


 if ( nargout==0 )
   clear fea out
 end
```

11.128 examples/ex_navierstokes13

EX_NAVIERSTOKES13 3D Example flow past a cylinder.

[FEA, OUT] =

EX_NAVIERSTOKES13(VARARGIN) Sets up and solves stationary and laminar 3D flow past a cylinder. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
```

```
rho         scalar {1}              Density
miu         scalar {0.001}          Molecular/dynamic viscosity
uin         scalar {0.45}           Magnitude of inlet velocity
sf_u        string {sf_hex_Q1nc}    Shape function for velocity
sf_p        string {sf_disc0}       Shape function for pressure
solver      string <tt>openfoam</tt>/{'} Use OpenFOAM or default solver
iplot       scalar 0/{1}            Plot solution and error (=1)
                                                                                .
Output      Value/(Size)            Description
--------------------------------------------------------------------------------
fea         struct                  Problem definition struct
out         struct                  Output struct
```

Code listing

```
 cOptDef = {   ...
   'rho',      1;
   'miu',      0.001;
   'uin',      0.45;
   'nlev',     1;
   'sf_u',     'sf_hex_Q1nc';
   'sf_p',     'sf_disc0';
   'tol',      0.2;
   'solver',   'openfoam';
   'iplot',    1;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry.
 fea.sdim = { 'x', 'y', 'z' };
 gobj1 = gobj_block( 0, 2.5, 0, 0.41, 0, 0.41, 'B1' );
 gobj2 = gobj_cylinder( [0.5 0 0.2], 0.05, 0.41, 2, 'C1' );
 fea.geom.objects = { gobj1 gobj2 };
 fea = geom_apply_formula( fea, 'B1-C1' );


% Grid generation.
 ns = 8*2^(opt.nlev-1);
 r  = [0.05 0.06 0.08 0.11 0.15];
 x  = [0.41 0.5 0.7 1 1.4 1.8 2.2] + 0.3;
 for ilev=2:opt.nlev
   r = sort( [ r (r(1:end-1)+r(2:end))/2 ] );
   x = sort( [ x (x(1:end-1)+x(2:end))/2 ] );
 end
```

```
 grid1 = ringgrid( r, 4*ns, [], [], [0.5;0.2] );
 grid2 = holegrid( ns, 2^(opt.nlev-1), [0.3 0.71;0 0.41], 0.15, [0.5;0.2] );
 grid2 = gridmerge( grid1, 5:8, grid2, 1:4 );
 grid3 = rectgrid( x, ns, [0.71 2.5;0 0.41] );
 fea.grid = gridmerge( grid3, 4, grid2, 6 );
 grid4 = rectgrid( 1, ns, [0 0.3;0 0.41] );
 fea.grid = gridmerge( fea.grid, 10, grid4, 2 );
 fea.grid = gridextrude( fea.grid, 5, 0.41, 2 );
 fea.grid = gridscale( fea.grid, [1 1 -1] );
 fea.grid.b(3,:) = 1;
 [~,ix] = findbdr( fea, 'x<=sqrt(eps)', 0 );
 fea.grid.b(3,ix) = 2;
 [~,ix] = findbdr( fea, 'x>=2.5-sqrt(eps)', 0 );
 fea.grid.b(3,ix) = 3;
 fea.grid.s(:) = 1;


% Problem definition.
 fea = addphys( fea, @navierstokes );
 fea.phys.ns.eqn.coef{1,end} = { opt.rho };
 fea.phys.ns.eqn.coef{2,end} = { optiu };
 fea.phys.ns.sfun            = { opt.sf_u opt.sf_u opt.sf_u opt.sf_p };


% Boundary conditions.
 fea.phys.ns.bdr.sel(2) = 2;
 fea.phys.ns.bdr.sel(3) = 4;
 if( strcmp(opt.solver,'openfoam') )
   fea.phys.ns.bdr.coef{2,end}{1,2} = 3/2*opt.uin;
 else
   fea.phys.ns.bdr.coef{2,end}{1,2} = ['16*',num2str(opt.uin),'*(y*z*(0.41-y)*(0.41-z))/0.41^4'];
 end


% Parse and solve problem.
 fea  = parsephys( fea );
 fea  = parseprob( fea );
 if( strcmp(opt.solver,'openfoam') )
   fea = openfoam( fea );
 else
   fea.sol.u = solvestat( fea, 'fid', fid );
 end


% Postprocessing.
 if( opt.iplot>0 )
   postplot( fea, 'sliceexpr', 'sqrt(u^2+v^2+w^2)' )
 end


% Error checking.
 s_tfx = ['nx*p+',num2str(optiu),'*(-2*nx*ux-nz*(uz+vx))'];
```

```
  s_cd  = ['-2*(',s_tfx,')/(',num2str(opt.rho),'*',num2str(0.2),'^2*',num2str(0.1*0.41),')'];
  c_d1  = intbdr(s_cd,fea,[7:10],10);
  dp    = evalexpr('p',[0.35 0.55;0.2 0.2;0.2 0.2],fea);
  out.err = [abs(c_d1-6.185333)/6.185333, ...
             abs(dp(1)-dp(2)-0.171007)/0.171007];
  out.pass = all(out.err < [0.15 0.6]);


  if ( nargout==0 )
    clear fea out
  end
```

11.129 examples/ex_navierstokes14

EX_NAVIERSTOKES14 2D Example for flow in a channel driven by pressure difference.

[FEA, OUT] =

EX_NAVIERSTOKES14(VARARGIN) Sets up and solves stationary Poiseuille flow in a rectangular channel driven by a pressure difference. The flow profile is constant and should assume a parabolic profile (u(y)=dp/dx/2/miu∗y∗(h-y)). Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {0.1}           Density
miu         scalar {0.2}           Molecular/dynamic viscosity
dp          scalar {0.3}           Pressure difference between in and out-flow
h           scalar {0.5}           Channel height
l           scalar {2.5}           Channel length
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
stper       scalar 1/{0}           Compute Stokes problem with periodic pressure
hmax        scalar {0.04}          Max grid cell size
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iphys       scalar 0/{1}           Use physics mode to define problem (=1)
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'rho',      0.1;
             'miu',      0.2;
             'dp',       0.3;
             'h',        0.5;
             'l',        2.5;
             'igrid',    1;
             'stper',    0;
             'hmax',     0.5/10;
             'sf_u',     'sflag2';
             'sf_p',     'sflag1';
             'iphys',    1;
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Model parameters.
 rho       = opt.rho;     % Density.
 miu       = optiu;     % Molecular/dynamic viscosity.
 global dp
 dp        = opt.dp;      % Pressure differetial.
% Geometry and grid parameters.
 h         = opt.h;       % Height of rectangular domain.
 l         = opt.l;       % Length of rectangular domain.
% Discretization parameters.
 sf_u      = opt.sf_u;    % FEM shape function type for velocity.
 sf_p      = opt.sf_p;    % FEM shape function type for pressure.


% Geometry definition.
 gobj = gobj_rectangle( 0, l, 0, h );
 fea.geom.objects = { gobj };
 fea.sdim = { 'x' 'y' };   % Coordinate names.


% Grid generation.
 fea.grid = rectgrid(round(l/opt.hmax),round(h/opt.hmax),[0 l;0 h]);
 if( opt.igrid~=0 )
   fea.grid = quad2tri( fea.grid );
 end


% Boundary conditions.
 dtol      = opt.hmax/2;
 i_inflow  = findbdr( fea, ['x<',num2str(dtol)] );     % Inflow boundary number.
 i_outflow = findbdr( fea, ['x>',num2str(l-dtol)] );   % Outflow boundary number.
```

```
% Problem definition.
 fea = addphys(fea,@navierstokes);        % Add Navier-Stokes equations physics mode.
 fea.phys.ns.eqn.coef{1,end} = { rho };
 fea.phys.ns.eqn.coef{2,end} = { miu };
 fea.phys.ns.sfun = { sf_u sf_u sf_p };   % Set shape functions.

 fea.phys.ns.bdr.sel([i_inflow i_outflow]) = 3;
 fea = parsephys(fea);   % Check and parse physics modes.

 [fea.bdr.n{1}{[i_inflow i_outflow]}] = deal( '-p*nx' );   % Offset natural Neumann BC.


% Parse and solve problem.
 if( opt.stper )
% Save pediodic boundary function to file.
   s_file = fileread( [mfilename('fullpath'),''] );
   ix = strfind( s_file, 'function' );
   s_fcn = s_file(ix(end):end);
   ix1 = find( s_fcn == '=', 1 );
   ix2 = find( s_fcn == '(', 1 );
   s_fname = strtrim(s_fcn(ix1+1:ix2-1));
   fid_tmp = fopen( [s_fname,''], 'w' );
   fprintf( fid_tmp, '%s', s_fcn );
   fclose( fid_tmp );

   [fea.bdr.n{3}{[i_inflow i_outflow]}] = deal( ['solve_hook_',s_fname] );   % Set periodic BCs.

   fea       = parseprob( fea );               % Check and parse problem struct.
   fea.sol.u = solvestat( fea, 'fid', fid, 'nlinasm', [1 1 1], 'tolchg', 1e-3 );
% Call to stationary solver.
 else
   fea.bdr.d{3}{i_inflow}  = dp;
   fea.bdr.d{3}{i_outflow} = 0;

   fea       = parseprob( fea );               % Check and parse problem struct.
   fea.sol.u = solvestat( fea, 'fid', fid );   % Call to stationary solver.
 end


% Postprocessing.
 s_velm = 'sqrt(u^2+v^2)';
 s_refsol = [num2str(dp),'/',num2str(l),'/2/',num2str(miu),'*y*(',num2str(h),'-y)'];
% Definition of velocity profile.
 p = [ l/2*ones(1,25); linspace(0,h,25) ];
 u = evalexpr( s_velm, p, fea );
 u_ref = evalexpr( s_refsol, p, fea );
 if ( opt.iplot>0 )
   figure
   subplot(3,1,1)
   postplot(fea,'surfexpr',s_velm,'evaltype','exact')
   title('Velocity field')
```

```
   subplot(3,1,2)
   postplot(fea,'surfexpr','p','evaltype','exact')
   title('Pressure')

   subplot(3,1,3)
   plot( u, p(2,:) )
   hold on
   plot( u_ref, p(2,:), 'k.' )
   legend( 'Computed solution', 'Reference solution','Location','West')
   title( 'Velocity profile at x=l/2' )
   ylabel( 'y' )
 end


% Error checking.
 err = sqrt(sum((u-u_ref).^2)/sum(u_ref.^2));
 if( ~isempty(fid) )
   fprintf(fid,'\nL2 Error: %f\n',err)
   fprintf(fid,'\n\n')
 end


 out.err  = err;
 out.pass = err<0.05;
 if ( nargout==0 )
   clear fea out
 end


%
```

11.130 examples/ex_swirl_flow1

EX_SWIRL_FLOW1 2D Axisymmetric laminar swirl flow.

[FEA, OUT] =

EX_SWIRL_FLOW1(VARARGIN) Axisymmetric swirl for in tubular region where the inner cylindrical wall is rotating. Comparison with analytical solution.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {2}             Density
miu         scalar {3}             Molecular/dynamic viscosity
```

```
omega       scalar {5}              Angular rotational frequency (of inner wall)
ri          scalar {0.5}            Inner radius
ro          scalar {1.5}            Outer radius
h           scalar {3}              Height of cylinder
sf_u        string {sflag2}         Shape function for velocity
sf_p        string {sflag1}         Shape function for pressure
iplot       scalar 0/{1}            Plot solution (=1)
                                                                                .
Output      Value/(Size)            Description
--------------------------------------------------------------------------------
fea         struct                  Problem definition struct
out         struct                  Output struct
```

Code listing

```
 cOptDef = { 'rho',      2;
             'miu',      3;
             'omega',    5;
             'ri',       0.5
             'ro',       1.5;
             'h',        3;
             'sf_u',     'sflag2';
             'sf_p',     'sflag1';
             'iplot',    1;
             'tol',      0.01;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = {'r' 'z'};

 ri = opt.ri;   % Inner radius.
 ro = opt.ro;   % Outer radius.
 h  = opt.h;    % Height of cylinder.
 fea.geom.objects = { gobj_rectangle(ri,ro,-h/2,h/2) };

 fea.grid = gridgen( fea, 'hmax', (ro-ri)/6, 'fid', fid );


% Equation definition.
 fea.dvar = { 'u', 'v', 'w', 'p' };
 fea.sfun = [ repmat( {opt.sf_u}, 1, 3 ) {opt.sf_p} ];
 c_eqn    = { 'r*rho*u'' - r*miu*(2*ur_r + uz_z  +   wr_z) + r*rho*(u*ur_t + w*uz_t) + r*p_r
= r*Fr - 2*miu/r*u_t + p_t + rho*v*v_t';
              'r*rho*v'' - r*miu*(  vr_r + vz_z) + miu*v_r + r*rho*(u*vr_t + w*vz_t) + rho*u*v_t =
```

```
              'r*rho*w'' - r*miu*(  wr_r + uz_r  + 2*wz_z) + r*rho*(u*wr_t + w*wz_t) + r*p_z
= r*Fz';
              'r*ur_t + r*wz_t + u_t = 0' };
 fea.eqn = parseeqn( c_eqn, fea.dvar, fea.sdim );

 fea.coef = { 'rho', opt.rho ;
              'miu', optiu ;
              'Fr',  0 ;
              'Ft',  0 ;
              'Fz',  0 };


% Boundary conditions.
 fea.bdr.d = { []  0 [] 0  ;
               []  0 [] opt.omega*ri ;
                0  0  0 [] ;
               [] [] [] [] };
 fea.bdr.n = cell(size(fea.bdr.d));


% Fix pressure at p([r,z]=[ro,h/2]) = 0.
 [~,ix_p] = min( sqrt( (fea.grid.p(1,:)-ro).^2 + (fea.grid.p(2,:)-h/2).^2) );
 fea.pnt = struct( 'type',  'constr', ...
                   'index', ix_p, ...
                   'dvar',  'p', ...
                   'expr',  '0' );


% Define analytical Newton Jacobian bilinear form.
 jac.coef  = { 'r*rho*ur'       'rho*v' 'r*rho*uz' [] ;
               'r*rho*vr+rho*v' []      'r*rho*vz' [] ;
               'r*rho*wr'       []      'r*rho*wz' [] ;
               []               []      []         [] };
 jac.form = cell(size(jac.coef));
 [jac.form{~cellfun(@isempty,jac.coef)}] = deal([1;1]);


% Parse and solve problem.
 fea = parseprob( fea );
 fea.sol.u = solvestat( fea, 'maxnit', 30, 'nlrlx', 1.0, 'nsolve', 2, 'jac', jac, 'fid', fid );


% Exact (analytical) solution.
 a = - opt.omega*ri^2 / (ro^2-ri^2);
 b =   opt.omega*ri^2*ro^2 / (ro^2-ri^2);
 v_th_ex = @(r,a,b) a.*r + b./r;


% Postprocessing.
 if( opt.iplot )
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'sqrt(u^2+v^2+w^2)', 'isoexpr', 'v' )
```

```
   subplot(1,2,2)
   hold on
   grid on
   r = linspace( ri, ro, 100 );
   v_th = evalexpr( 'v', [r;zeros(1,length(r))], fea );
   plot( r, v_th, 'b--' )
   r = linspace( ri, ro, 10 );
   plot( r, v_th_ex(r,a,b), 'r.' )
   legend( 'Computed solution', 'Exact solution')
   xlabel( 'Radius, r')
   ylabel( 'Angular velocity, v')
 end


% Error checking.
 r = linspace( ri, ro, 100 );
 v_th = evalexpr( 'v', [r;zeros(1,length(r))], fea )';
 out.err  = norm( v_th - v_th_ex(r,a,b) );
 out.pass = out.err < opt.tol;


 if( nargout==0 )
   clear fea out
 end
```

11.131 examples/ex_swirl_flow2

EX_SWIRL_FLOW2 2D Axisymmetric swirl flow in step domain.

[FEA, OUT] =

EX_SWIRL_FLOW2(VARARGIN) Axisymmetric swirl for in tubular step region where the inner cylindrical wall is rotating.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
omega       scalar {100}           Angular rotational velocity (of inner wall)
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
```

```
out         struct                  Output struct
```

Code listing

```
 cOptDef = { 'omega',    100;
             'sf_u',     'sflag2';
             'sf_p',     'sflag1';
             'iplot',    1;
             'tol',      0.05;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = {'r' 'z'};

 fea.geom.objects = { gobj_rectangle(0.5,1.5,0,3) gobj_rectangle(1.0,1.5,1.5,3,'R2') };
 fea = geom_apply_formula( fea, 'R1-R2' );

 fea.grid = gridgen( fea, 'hmax', 0.1, 'fid', fid );


% Equation definition.
 fea.dvar = { 'u', 'v', 'w', 'p' };
 fea.sfun = [ repmat( {opt.sf_u}, 1, 3 ) {opt.sf_p} ];
 c_eqn    = { 'r*rho*u'' - r*miu*(2*ur_r + uz_z  +   wr_z) + r*rho*(u*ur_t + w*uz_t) + r*p_r
= r*Fr - 2*miu/r*u_t + p_t + rho*v*v_t';
             'r*rho*v'' - r*miu*(  vr_r + vz_z) + miu*v_r + r*rho*(u*vr_t + w*vz_t) + rho*u*v_t =
             'r*rho*w'' - r*miu*(  wr_r + uz_r  + 2*wz_z) + r*rho*(u*wr_t + w*wz_t) + r*p_z
= r*Fz';
             'r*ur_t + r*wz_t + u_t = 0' };
 fea.eqn = parseeqn( c_eqn, fea.dvar, fea.sdim );

 fea.coef = { 'rho', 1 ;
              'miu', 1 ;
              'Fr',  0 ;
              'Ft',  0 ;
              'Fz',  0 };


% Boundary conditions.
 fea.bdr.d = { 0  0  [] 0  0  0 ;
               0  0  [] opt.omega 0 0 ;
               0  0  0  0  0  0 ;
               [] [] [] [] [] [] };
 fea.bdr.n = cell(size(fea.bdr.d));
```

```
% Fix pressure at p([r,z]=[ro,h/2]) = 0.
 [~,ix_p] = min( sqrt( (fea.grid.p(1,:)-1.5).^2 + (fea.grid.p(2,:)-1.5).^2) );
 fea.pnt = struct( 'type',  'constr', ...
                   'index', ix_p, ...
                   'dvar',  'p', ...
                   'expr',  '0' );


% Define analytical Newton Jacobian bilinear form.
 jac.coef  = { 'r*rho*ur'       'rho*v' 'r*rho*uz' [] ;
               'r*rho*vr+rho*v' []      'r*rho*vz' [] ;
               'r*rho*wr'       []      'r*rho*wz' [] ;
               []               []      []         [] };
 jac.form = cell(size(jac.coef));
 [jac.form{~cellfun(@isempty,jac.coef)}] = deal([1;1]);


% Parse and solve problem.
 fea = parseprob( fea );
 fea.sol.u = solvestat( fea, 'maxnit', 50, 'nlrlx', 1.0, 'nsolve', 2, 'jac', jac, 'fid', fid );


% Postprocessing.
 if( opt.iplot )
   postplot( fea, 'surfexpr', 'sqrt(u^2+v^2+w^2)', 'isoexpr', 'v' )
 end


% Error checking.
 out.ref  = [ -6.1 10.5 73 1.25 ];
 [u_min,u_max] = minmaxsubd( 'u', fea );
 out.err  = [ u_min u_max intsubd('v',fea) intsubd('w',fea) ];
 out.pass = mean(abs(out.err-out.ref)./abs(out.ref)) < opt.tol;

 if( nargout==0 )
   clear fea out
 end
```

11.132 examples/ex_swirl_flow3

EX_SWIRL_FLOW3 2D Axisymmetric Taylor-Couette (swirl) flow.

[FEA, OUT] =

EX_SWIRL_FLOW3(VARARGIN) Axisymmetric Taylor-Couette swirl flow in a tubular region where the inner cylindrical wall is rotating. Time dependent solution with periodic top and

bottom.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
rho         scalar {1}             Density
miu         scalar {1}             Molecular/dynamic viscosity
omega       scalar {300}           Maximum angular velocity (of inner wall)
tmax        scalar {3}             Maximum time
nstep       scalar {300}           Number of time steps
ri          scalar {1.0}           Inner radius
ro          scalar {1.5}           Outer radius
h           scalar {3}             Height of cylinder
sf_u        string {sflag2}        Shape fcn for velocity
sf_p        string {sflag1}        Shape fcn for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'rho',      1;
             'miu',      1;
             'omega',    300;
             'tmax',     3;
             'nstep',    300;
             'ri',       1.0
             'ro',       1.5;
             'h',        3;
             'sf_u',     'sflag2';
             'sf_p',     'sflag1';
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = {'r' 'z'};

 ri = opt.ri;   % Inner radius.
 ro = opt.ro;   % Outer radius.
 h  = opt.h;    % Height of cylinder.
```

```
 fea.geom.objects = { gobj_rectangle(ri,ro,-h/2,h/2) };

 n = 16;
 fea.grid = rectgrid( n, 6*n, [ri -h/2 ro h/2] );


% Equation definition.
 fea.dvar = { 'u', 'v', 'w', 'p' };
 fea.sfun = [ repmat( {opt.sf_u}, 1, 3 ) {opt.sf_p} ];
 c_eqn    = { 'r*rho*u'' - r*miu*(2*ur_r + uz_z  +   wr_z) + r*rho*(u*ur_t + w*uz_t) + r*p_r
= r*Fr - 2*miu/r*u_t + p_t + rho*v*v_t';
              'r*rho*v'' - r*miu*(  vr_r + vz_z) + miu*v_r + r*rho*(u*vr_t + w*vz_t) + rho*u*v_t =
              'r*rho*w'' - r*miu*(  wr_r + uz_r  + 2*wz_z) + r*rho*(u*wr_t + w*wz_t) + r*p_z
= r*Fz';
              'r*ur_t + r*wz_t + u_t = 0' };

 fea.eqn = parseeqn( c_eqn, fea.dvar, fea.sdim );

 fea.coef = { 'rho', opt.rho ;
              'miu', optiu ;
              'Fr',  0 ;
              'Ft',  0 ;
              'Fz',  0 };


% Boundary conditions.
 fea.bdr.d = { []  0 [] 0 ;
               []  0 [] sprintf( '%g*t/%g', opt.omega*ri, opt.tmax ) ;
               []  0 [] 0 ;
               [] [] [] [] };
 fea.bdr.n = cell(size(fea.bdr.d));


% Save pediodic boundary function to file.
 s_file = fileread( [mfilename('fullpath'),''] );
 ix = strfind( s_file, 'function' );
 s_fcn = s_file(ix(end):end);
 ix1 = find( s_fcn == '=', 1 );
 ix2 = find( s_fcn == '(', 1 );
 s_fname = strtrim(s_fcn(ix1+1:ix2-1));
 fid_tmp = fopen( [s_fname,''], 'w' );
 fprintf( fid_tmp, '%s', s_fcn );
 fclose( fid_tmp );

 [fea.bdr.n{1:3,1}] = deal( ['solve_hook_',s_fname] );   % Set periodic BCs.


% Fix pressure at p([r,z]=[ro,0]) = 0.
 [~,ix_p] = min( sqrt( (fea.grid.p(1,:)-ro).^2 + (fea.grid.p(2,:)-0).^2) );
 fea.pnt = struct( 'type',  'constr', ...
                   'index', ix_p, ...
                   'dvar',  'p', ...
```

```
                    'expr',  '0' );


% Parse and solve problem.
 fea = parseprob( fea );
 fea.sol.u = solvetime( fea, 'ischeme', 1, 'tstep', opt.tmax/opt.nstep, 'tmax', opt.tmax, 'tolchg'
 delete( [s_fname,''] )


% Postprocessing.
 if( opt.iplot )
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'sqrt(u^2+w^2)', 'isoexpr', 'sqrt(u^2+w^2)', ...
                  'arrowexpr', {'u' 'w'}, 'arrowcolor', 'w', 'arrowspacing', [8 48] )
   title('In-plane velocity')

   subplot(1,2,2)
   postplot( fea, 'surfexpr', 'v', 'isoexpr', 'v' )
   title('Azimuthal velocity')
 end


 out = [];
 if( nargout==0 )
   clear fea out
 end


%
```

11.133 examples/ex_swirl_flow4

EX_SWIRL_FLOW4 2D Rotating swirling flow around a disk.

[FEA, OUT] =

EX_SWIRL_FLOW4(VARARGIN) Axisymmetric swirling flow around a rotating disk immersed in a container.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
sf_u        string {sflag2}        Shape function for velocity
sf_p        string {sflag1}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
```

```
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'sf_u',     'sflag2';
             'sf_p',     'sflag1';
             'iplot',    1;
             'tol',      0.01;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = {'r' 'z'};

 fea.geom.objects = { gobj_polygon([0 0;.03 0;.03 .05;.002 .05;.002 .02;.008 .018;.0085 .013;0 .01


 hmax = 2e-3;
 try
   hmaxe = 4e-4;
   fea.grid = gridgen_triangle( fea, 'hmax', hmax, 'hmaxe', hmaxe, 'fid', fid );
 catch
   fea.grid = gridgen( fea, 'hmax', hmax, 'fid', fid );
 end


% Equation definition.
 fea.dvar = { 'u', 'v', 'w', 'p' };
 fea.sfun = [ repmat( {opt.sf_u}, 1, 3 ) {opt.sf_p} ];
 c_eqn    = { 'r*rho*u'' - r*miu*(2*ur_r + uz_z  +   wr_z) + r*rho*(u*ur_t + w*uz_t) + r*p_r
= r*Fr - 2*miu/r*u_t + p_t + rho*v*v_t';
              'r*rho*v'' - r*miu*(  vr_r + vz_z) + miu*v_r + r*rho*(u*vr_t + w*vz_t) + rho*u*v_t =
              'r*rho*w'' - r*miu*(  wr_r + uz_r  + 2*wz_z) + r*rho*(u*wr_t + w*wz_t) + r*p_z
= r*Fz';
              'r*ur_t + r*wz_t + u_t = 0' };
 fea.eqn = parseeqn( c_eqn, fea.dvar, fea.sdim );

 fea.coef = { 'rho', 1.2e3 ;
              'miu', 2.3e-3 ;
              'Fr',  0 ;
              'Ft',  0 ;
              'Fz',  0 };
```

```
% Boundary conditions.
 n_bdr = max(fea.grid.b(3,:));
 fea.bdr.d = cell(4,n_bdr);
 [fea.bdr.d{1,[1 2 5:7 8]}] = deal(0);
 [fea.bdr.d{2,[1 2]}] = deal(0);
 [fea.bdr.d{3,[1 2 3 4:7]}] = deal(0);
 [fea.bdr.d{2,4:7}] = deal('r*pi/6');
 [fea.bdr.d{1:3,3}] = deal([]);
 fea.bdr.n = cell(size(fea.bdr.d));


% Fix pressure at p([r,z]=[0.03,0.05]) = 0.
 [~,ix_p] = min( sqrt( (fea.grid.p(1,:)-0.03).^2 + (fea.grid.p(2,:)-0.05).^2) );
 fea.pnt = struct( 'type',  'constr', ...
                   'index', ix_p, ...
                   'dvar',  'p', ...
                   'expr',  '0' );


% Define analytical Newton Jacobian bilinear form.
 jac.coef  = { 'r*rho*ur'       'rho*v' 'r*rho*uz' [] ;
               'r*rho*vr+rho*v' []      'r*rho*vz' [] ;
               'r*rho*wr'       []      'r*rho*wz' [] ;
               []               []      []         [] };
 jac.form = cell(size(jac.coef));
 [jac.form{~cellfun(@isempty,jac.coef)}] = deal([1;1]);


% Parse and solve problem.
 fea = parseprob( fea );
 fea.sol.u = solvestat( fea, 'jac', jac, 'fid', fid, 'nsolve', 2, 'jac', jac );


% Postprocessing.
 if( opt.iplot )
   postplot( fea, 'surfexpr', 'sqrt(u^2+v^2+w^2)', 'isoexpr', 'v', 'isolev', 15, 'isocolor', 'w' )
 end


% Error checking.
 [~,U_max] = minmaxsubd( 'sqrt(u^2+v^2+w^2)', fea );
 out.err  = norm( U_max - 4.445e-3 )/4.445e-3;
 out.pass = out.err < opt.tol;


 if( nargout==0 )
   clear fea out
 end
```

11.134 examples/ex_piezoelectric1

EX_PIEZOELECTRIC1 Piezoelectric bending of a beam example.

[FEA, OUT] =

EX_PIEZOELECTRIC1(VARARGIN) Example for piezoelectric bending of a beam. Reference: Hwang, W. S., Park, H. C. Finite Element Modeling of Piezoelectric Sensors and Actuators. AIAA Journal, 31(5), pp 930-937, 1993.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
l           scalar {0.12}          Length of beam
h           scalar {2e-3}          Height (thickness) of beam
delV        scalar {200}           Potential difference
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {20}            Cell resolution
sfund       string {sflag1}        Shape function for displacements
sfunp       string {sflag1}        Shape function for potential
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'l',        0.12; ...
   'h',        2e-3; ...
   'delV',     100; ...
   'igrid',    0; ...
   'hmax',     20; ...
   'sfund',    'sflag2'; ...
   'sfunp',    'sflag1'; ...
   'iplot',    1; ...
   'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;
 i_impl    = 1;

% Geometry definition.
 fea.sdim = { 'x' 'y' };   % Names of space coordinates.
```

```
 gobj1    = gobj_rectangle( 0, opt.l,  0,       opt.h/2, 'R1' );
 gobj2    = gobj_rectangle( 0, opt.l, -opt.h/2, 0,       'R2' );
 fea.geom.objects = { gobj1 gobj2 };


% Grid generation.
 switch opt.igrid
   case -1
     fea.grid = rectgrid( opt.hmax, opt.hmax, [ 0 opt.l; -opt.h/2 opt.h/2] );
     fea.grid = quad2tri( fea.grid );
   case 0
     fea.grid = rectgrid( opt.hmax, opt.hmax, [ 0 opt.l; -opt.h/2 opt.h/2] );
     fea.grid.s( selcells( fea, 'y<=0' ) ) = 2;
   case 1
     fea.grid = gridgen( fea, 'hmax', opt.h/(opt.hmax/4.6), 'fid', fid );
     fea.grid.s(:) = 1;
 end
 fea.grid.s( selcells( fea, 'y<=0' ) ) = 2;


% Equation coefficients.
 Emod =  2e9;       % Modulus of elasticity
 nu   =  0.29;      % Poissons ratio
 Gmod =  0.775e9;   % Shear modulus
 d31  =  0.22e-10;  % Piezoelectric sn coefficient
 d33  = -0.3e-10;   % Piezoelectric sn coefficient
 prel = 12;         % Relative electrical permittivity
 pvac = 0.885418781762-11;   % Electrical permittivity of vacuum

% Constitutive relations.
 constrel   = [ Emod/(1-nu^2)    nu*Emod/(1-nu^2) 0    ;
                nu*Emod/(1-nu^2) Emod/(1-nu^2)    0    ;
                0                0                Gmod ];
 piezoel_st = [ 0 d31 ;
                0 d33 ;
                0 0   ];
 piezoel_sn = constrel*piezoel_st;
 dielmat_st = [ prel 0    ;
                0    prel ]*pvac ;
 dielmat_sn = [ dielmat_st - piezoel_st'*piezoel_sn ];

% Populate coefficient matrices (negative sign due to fem partial integration).
 c{1,1} = { constrel(1,1)   constrel(1,3) ;
            constrel(1,3)   constrel(3,3) };
 c{1,2} = { constrel(1,3)   constrel(1,2) ;
            constrel(3,3)   constrel(2,3) };
 c{2,1} = c{1,2}';
 c{2,2} = { constrel(3,3)   constrel(1,3) ;
            constrel(1,3)   constrel(2,2) };
 c{1,3} = { piezoel_sn(1,1) piezoel_sn(1,2) ;
            piezoel_sn(3,1) piezoel_sn(3,2) };
 if( i_impl )
```

```
    for i=1:4
      c{1,3}{i} = [num2str(c{1,3}{i}),'*(2*(y<0)-1)'];
    end
  else
    for i=1:4
      fea.expr{i,1} = ['piezoel_sn1',num2str(i)];
      fea.expr{i,2} = { -c{1,3}{i} c{1,3}{i} };
      c{1,3}{i} = ['piezoel_sn1',num2str(i)];
    end
  end
  c{3,1} = c{1,3}';
  c{2,3} = { piezoel_sn(3,1) piezoel_sn(3,2);
             piezoel_sn(2,1) piezoel_sn(2,2) };
  if( i_impl )
    for i=1:4
      c{2,3}{i} = [num2str(c{2,3}{i}),'*(2*(y<0)-1)'];
    end
  else
    for i=1:4
      fea.expr{i+4,1} = ['piezoel_sn2',num2str(i)];
      fea.expr{i+4,2} = { -c{2,3}{i} c{2,3}{i} };
      c{2,3}{i} = ['piezoel_sn2',num2str(i)];
    end
  end
  c{3,2} = c{2,3}';
  c{3,3} = { dielmat_sn(1,1) dielmat_sn(2,1) ;
             dielmat_sn(2,1) dielmat_sn(2,2) };


% Dependent variable names.
 fea.dvar  = { 'u' 'v' 'V' };
 n_dvar    = length(fea.dvar);

% Finite element shape functions.
 fea.sfun  = { opt.sfund opt.sfund opt.sfunp };

% Define equations.
 bilinear_form = [ 2 2 3 3 ;
                   2 3 2 3 ];
 for i=1:n_dvar
   for j=1:n_dvar
     fea.eqn.a.form{i,j} = bilinear_form;
     fea.eqn.a.coef{i,j} = c{i,j}(:)';
   end
 end

% Source terms (set to zero).
 fea.eqn.f.form = { 1 1 1 };
 fea.eqn.f.coef = { 0 0 0 };


% Boundary conditions.
```

```
 n_bdr          = max(fea.grid.b(3,:));   % Number of boundaries.
 fea.bdr.d      = cell(n_dvar,n_bdr);
 fea.bdr.n      = cell(n_dvar,n_bdr);
 i_top          = findbdr( fea, ['y>=',num2str(opt.h/2-sqrt(eps))] );
 i_bottom       = findbdr( fea, ['y<=',num2str(-opt.h/2+sqrt(eps))] );
 i_left         = findbdr( fea, ['x<=',num2str(sqrt(eps))] );
 [fea.bdr.d{3,i_top}]    = deal( opt.delV );   % Set potential to dV on top boundary.
 [fea.bdr.d{3,i_bottom}] = deal( 0 );          % Set potential to 0V on bottom boundary.
 [fea.bdr.d{1:2,i_left}] = deal( 0 );          % Set displacements to 0 on left boundary.


% Check and parse problem struct.
 fea = parseprob( fea );

% Call to stationary solver.
 fea.sol.u = solvestat( fea, 'fid', fid );


% Postprocessing.
 if( opt.iplot )
   YSCALE = 3;
   axlim  = [0, opt.l, -YSCALE*opt.h/2, YSCALE*opt.h/2];
   DSCALE = 20;

   subplot(2,2,1)
   postplot( fea, 'surfexpr', 'u', 'isoexpr', 'u', 'setaxes', 'off' )
   axis(axlim);
   title('x-displacement')

   subplot(2,2,2)
   postplot( fea, 'surfexpr', 'v', 'isoexpr', 'v', 'setaxes', 'off' )
   axis(axlim);
   title('y-displacement')

   subplot(2,2,3)
   postplot( fea, 'surfexpr', 'V', 'isoexpr', 'V', 'setaxes', 'off' )
   axis(axlim);
   title('Electric potential')

   subplot(2,2,4)
   plotsubd( fea, 'labels', 'off', 'setaxes', 'off' )
   axis(axlim);
   title(['Displacement plot (at ',num2str(DSCALE),' times scale)'])

   ind1 = sub2ind( size(fea.grid.c), fea.grid.b(2,:), fea.grid.b(1,:) );
   p1b  = fea.grid.p( :, fea.grid.c(ind1) );
   ind2 = sub2ind( size(fea.grid.c), mod(fea.grid.b(2,:),size(fea.grid.c,1))+1, fea.grid.b(1,:) );
   p2b  = fea.grid.p( :, fea.grid.c(ind2) );
   up1  = DSCALE*evalexpr( 'u', p1b, fea );
   vp1  = DSCALE*evalexpr( 'v', p1b, fea );
   up2  = DSCALE*evalexpr( 'u', p2b, fea );
   vp2  = DSCALE*evalexpr( 'v', p2b, fea );
```

```
    hold on
    for i=1:size(p1b,2)
      plot( [p1b(1,i)+up1(i) p2b(1,i)+up2(i)], [p1b(2,i)+vp1(i) p2b(2,i)+vp2(i)], '-b', 'linewidth'
    end
    set( gca, 'ytick', [] )
  end


% Error checking.
  ind_dof_v  = fea.eqn.dofm{2}(:) + fea.eqn.ndof(1);
  out.v_max  = max(abs( fea.sol.u(ind_dof_v) ));
  v_max_ref  = abs(-3/2*d31*opt.delV*(opt.l/opt.h)^2);
  out.err    = abs(v_max_ref - out.v_max)/v_max_ref;
  out.pass   = out.err <= 0.1;

  if ( nargout==0 )
    clear fea out
  end
```

11.135 examples/ex_electrostatics1

EX_ELECTROSTATICS1 Electrostatic test example.

[FEA, OUT] =
EX_ELECTROSTATICS1(VARARGIN) Electrostatics test example.
Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
sfun        string {sflag2}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
  cOptDef = { 'sfun',     'sflag2';
              'iplot',    1;
```

```
              'tol',      5e-3;
              'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' };
 fea.grid = rectgrid(10);


% Problem definition.
 fea = addphys( fea, @electrostatics );
 fea.phys.es.eqn.coef{1,end} = { 2 };
 fea.phys.es.eqn.coef{2,end} = { 3 };
 fea.phys.es.eqn.coef{3,end} = { 4 };
 fea.phys.es.eqn.coef{4,end} = { 5 };
 fea.phys.es.sfun            = { opt.sfun };

 fea.phys.es.bdr.sel = [2 3 3 4];
 fea.phys.es.bdr.coef{3,end}{1,2} =  1;
 fea.phys.es.bdr.coef{3,end}{1,3} = -3;

% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'fid', opt.fid );   % Call to stationary solver.


% Postprocessing.
 if( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', 'V' )
   title( 'Electric potential' )
 end


% Error checking.
 V = intsubd( 'V', fea );
 D = intbdr( fea.phys.es.bdr.vars{2,2}, fea, 1:4 );
 out.err = [ abs(V-1.25)/1.25;
             abs(D+5)/5  ];
 out.pass = all( out.err < opt.tol );


 if ( nargout==0 )
   clear fea out
 end
```

11.136 examples/ex_electrostatics2

EX_ELECTROSTATICS2 Axisymmetric model of a spherical capacitor.

[FEA, OUT] =

EX_ELECTROSTATICS2(VARARGIN) Axisymmetric model of a spherical capacitor compared with an analytical solution.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
sfun        string {sflag2}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'sfun',     'sflag2';
             'iplot',    1;
             'tol',      5e-2;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Model constants and parameters.
 r1   = 0.003;
 r2   = 0.01;
 r3   = 0.012;
 eps0 = 8.85e-12;
 epsr = 3.9;
 q0   = 6e-11;


% Geometry and grid generation.
 fea.sdim = { 'r' 'z' };
 fea.geom.objects = { gobj_circle([0 0],r1,'C1') ...
                      gobj_circle([0 0],r2,'C2') ...
                      gobj_circle([0 0],r3,'C3') ...
                      gobj_rectangle(-1.1*r1,0,-1.1*r3,1.1*r3,'R1') ...
```

```
                      gobj_rectangle(-1.1*r2,0,-1.1*r3,1.1*r3,'R2') ...
                      gobj_rectangle(-1.1*r3,0,-1.1*r3,1.1*r3,'R3') };
 fea = geom_apply_formula( fea, 'C1-R1' );
 fea = geom_apply_formula( fea, 'C2-R2' );
 fea = geom_apply_formula( fea, 'C3-R3' );

 geom_fix = false;
 if( geom_fix )   % Manual geometry fix.
   fea.geom.objects = [ fea.geom.objects, fea.geom.objects(2) ];
   fea.geom.objects = [ fea.geom.objects, fea.geom.objects(1) ];
   fea.geom.objects{4}.tag = 'CS4';
   fea.geom.objects{5}.tag = 'CS5';
   fea = geom_apply_formula( fea, 'CS2-CS1' );
   fea = geom_apply_formula( fea, 'CS3-CS4' );
 end

 fea.grid = gridgen( fea, 'hmax', (r3-r2)/2, 'fid', opt.fid );


% Problem definition.
 if( geom_fix )
   fea.const = { 'sigma'  { 6e7     0      6e7   } ;
                 'eps0'   { eps0    eps0   eps0  } ;
                 'epsr'   { 1       epsr   1     } ;
                 'tscale' { 1e-17   1e-17  1e-17 } ;
                 'rho'    { q0*3/4/pi/(r1^3) 0 -q0*3/4/pi/(r3^3-r2^3) } };
 else
   fea.const = { 'sigma'  { 0      6e7   6e7   } ;
                 'eps0'   { eps0   eps0  eps0  } ;
                 'epsr'   { epsr   1     1     } ;
                 'tscale' { 1e-17  1e-17 1e-17 } ;
                 'rho'    { 0 q0*3/4/pi/(r1^3) -q0*3/4/pi/(r3^3-r2^3) } };
 end

 fea = addphys( fea, {@electrostatics 1} );
 fea.phys.es.eqn.coef{1,end} = { 'sigma+epsr*eps0/tscale' };
 fea.phys.es.eqn.coef{4,end} = { 'rho/tscale' };
 fea.phys.es.sfun            = { opt.sfun };
 n_bdr = max(fea.grid.b(3,:));
 fea.phys.es.bdr.sel = 4*ones(1,n_bdr);
 fea.phys.es.bdr.sel(findbdr(fea,['sqrt(r.^2+z.^2)>',num2str((r2+r3)/2)])) = 2;


% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'fid', opt.fid );   % Call to stationary solver.


% Postprocessing.
 if( opt.iplot>0 )
   postplot( fea, 'surfexpr', 'sigma*sqrt(Vr^2+Vz^2)', 'isoexpr', 'V', 'isocolor', 'w' )
```

```
    title( 'Surface: current density, contour: electric potential' )
  end


% Error checking.
  [Vmin,Vmax] = minmaxsubd( 'V', fea );
  eeng = intsubd( 'eps0*epsr*(Vr^2+Vz^2)*pi*r', fea );

  cap1 = q0/( Vmax - Vmin );
  cap2 = q0^2/(2*eeng);
  cap_ref = 4*pi*epsr*eps0/( 1/r1 - 1/r2 );

  out.err  = [ abs(cap_ref-cap1)/cap_ref ...
               abs(cap_ref-cap2)/cap_ref ];
  out.pass = all(out.err<opt.tol);


  if ( nargout==0 )
    clear fea out
  end
```

11.137 examples/ex_resistive_heating1

EX_RESISTIVE_HEATING1 Resistive heating of a tungsten filament.

[FEA, OUT] =

EX_RESISTIVE_HEATING1(VARARGIN) Example to calculate temperature generated by resistive heating of a tungsten spiral filament, such as for example in an incandescent bulb. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
ibc         scalar 1{2}            Boundary type (1=insulation, 2=radiation)
isolve      scalar 1{2}            Solve type (1=stat.coupled, 2=split quasistatic)
sfun        string {sflag1}        Shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                               .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'ibc',      2;
             'isolve',   2;
             'sfun',     'sflag1';
             'iplot',    1;
             'tol',      1e-2;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Set up geometry and grid.
 r_wire   = 0.0005;
 l_spiral = 0.005;
 r_spiral = 0.004;

 fea.grid = gridrotate( gridrevolve( circgrid( 2, 3, r_wire ), 80, l_spiral, 3, r_spiral ), -pi/2,
 fea.sdim = { 'x' 'y' 'z' };


% Problem constants and definitions.
 V0    = 0.2;         % Voltage difference.
 sigma = 1/52.8e-9;   % Electrical conductivity [1/Ohm/m].

 rho   = 19.25e3;     % Density [kg/m3].
 cp    = 133.9776;    % Specific heat capacity [J/kg/K].
 k     = 173;         % Thermal conductivity [W/m/K].
 if( opt.isolve==1 )
   Q   = [num2str(sigma),'*(Vx^2+Vy^2+Vz^2)'];   % Resistive heating source term.
 else
   Q   = 'qfunction'; % Resistive heating source term function.
 end

 Const = 5.670367e-8;       % Constant for radiation BC (Stefan-Bolzmann constant).
 T0    = 20+273.15;         % Initial temperature.
 Tamb  = 80+273.15;         % Mean ambient temperature.

 icub  = str2num(opt.sfun(end))+1; % Numerical quadrature order.
 tmax  = 20;                % Maximum time for temperature simulation.
 dt    = tmax/20;           % Time step size.


% Set up and solve electrostatics problem.
 feaV = fea;
 feaV = addphys( feaV, @conductivemediadc );
 feaV.phys.dc.sfun = { opt.sfun };

 feaV.phys.dc.eqn.coef{2,end} = {sigma};    % Electrical conductivity.
```

```
  feaV.phys.dc.bdr.sel([1 98])    = 1;       % Use Dirichlet / Electric potential BCs for end bound
  feaV.phys.dc.bdr.coef{1,end}{1} = V0;      % Set electric potential on boundary 1 to V0 (boundary

  if( opt.isolve==2 )
    feaV = parsephys( feaV );
    feaV = parseprob( feaV );
    u_V  = solvestat( feaV, 'fid', opt.fid );
  end

% Set up and solve heat transfer problem.
  fea = addphys( fea, @heattransfer );
  fea.phys.ht.sfun = { opt.sfun };

  fea.phys.ht.eqn.coef{1,end} = {rho};       % Density.
  fea.phys.ht.eqn.coef{2,end} = {cp};        % Heat capacity.
  fea.phys.ht.eqn.coef{3,end} = {k};         % Thermal conductivity.
  fea.phys.ht.eqn.coef{7,end} = {Q};         % Heat source term.

  if( opt.ibc==1 )
    fea.phys.ht.bdr.sel([2:97]) = deal(3);     % Use insulation flux boundary condition for all bou
  else
    fea.phys.ht.bdr.sel([2:97]) = deal(4);     % Use generalized heat flux boundary condition for a
  [fea.phys.ht.bdr.coef{4,end}{2:97}] = deal({0 0 0 Const Tamb});
  end
  fea.phys.ht.bdr.sel([1 98]) = deal(1);     % Prescribe fixed ambient temperature at the end point
  fea.phys.ht.bdr.coef{1,end}{ 1} = T0;
  fea.phys.ht.bdr.coef{1,end}{98} = T0;

  if( opt.isolve==2 )
% Add sigma and electric potential solution used by qfunction.
    fea.expr = { 's_sigma' '' '' sigma ;
                 'u_V'     '' '' u_V   };

    fea  = parsephys( fea );
    fea  = parseprob( fea );
    l_write_qfunction()
    u_T  = solvetime( fea, 'icub', icub, 'init', {T0}, 'tstep', dt, 'tmax', tmax, 'fid', opt.fid );
    delete( 'qfunction' )
  end


% Merge solutions.
  fea.phys.dc = feaV.phys.dc;
  if( isfield(fea,'dvar') )
    fea = rmfield( fea, 'dvar' );
    fea = rmfield( fea, 'eqn'  );
  end
  fea = parsephys( fea );
  fea = parseprob( fea );
  if( opt.isolve==1 )
    fea.sol.u = solvestat( fea, 'icub', icub, 'fid', opt.fid, 'nlrlx', 1-0.3*(opt.ibc==2) );
    u_T = fea.sol.u(1:fea.eqn.ndof(1),:);
```

```
 else
   fea.sol.u = [ u_T; repmat(u_V,1,size(u_T,2)) ];
 end


% Postprocessing.
 if( opt.iplot>0 )
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'V' )
   title( 'Electric potential, V')

   subplot(1,2,2)
   postplot( fea, 'surfexpr', 'T-273.15' )
   title( 'Temperature, T [C]')
 end


% Error checking.
 if( opt.isolve==1 )
   if( opt.ibc==1 )
     T_max_ref = 840;
   else
     T_max_ref = 688;
   end
 else
   if( opt.ibc==1 )
     T_max_ref = 787;
   else
     T_max_ref = 680;
   end
 end
 out.err  = abs( max(u_T(:)) - T_max_ref )/T_max_ref;
 out.pass = out.err < opt.tol;


 if( nargout==0 )
   clear fea out
 end



%------------------------------
```

11.138 examples/ex_magnetostatics1

EX_MAGNETOSTATICS1 Magnetostatic test example.

[FEA, OUT] =

EX_MAGNETOSTATICS1(VARARGIN) Magnetostatics test example.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
sfun        string {sflag2}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'sfun',     'sflag2';
             'iplot',    1;
             'tol',      1e-1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' };
 fea.grid = rectgrid(10);


% Problem definition.
 fea = addphys( fea, @magnetostatics );
 fea.physs.eqn.coef{1,end} = {  1 };
 fea.physs.eqn.coef{2,end} = {  2 };
 fea.physs.eqn.coef{3,end} = {  3 };
 fea.physs.eqn.coef{4,end} = { -4 };
 fea.physs.sfun            = { opt.sfun };

 fea.physs.bdr.sel = [2 4 3 1];
 fea.physs.bdr.coef{3,end}{1,3} = 6;
 fea.physs.bdr.coef{1,end}{1,4} = 'y';


% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'fid', opt.fid );   % Call to stationary solver.
```

```
% Postprocessing.
 if( opt.iplot>0 )
   figure
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'Az', 'arrowexpr', fea.physs.eqn.vars{8,2} )
   title( 'Magnetic potential' )
   subplot(1,2,2)
   postplot( fea, 'surfexpr', fea.physs.eqn.vars{5,2}, 'arrowexpr', fea.physs.eqn.vars{9,2} )
   title( 'Magnetic field' )
 end


% Error checking.
 Az  = intsubd( 'Az', fea );
 Mf  = intsubd( fea.physs.eqn.vars{2,2}, fea );
 Scb = intbdr(  fea.physs.bdr.vars{2,2}, fea, 1:4 );
 out.err = [ abs(Az-3.21)/3.21  ;
             abs(Mf-4.9)/4.9    ;
             abs(Scb+2.02)/2.02 ];
 out.pass = all( out.err < opt.tol );


 if ( nargout==0 )
   clear fea out
 end
```

11.139 examples/ex_magnetostatics2

EX_MAGNETOSTATICS2 Magnetic field around a horseshoe magnet.

[FEA, OUT] =
EX_MAGNETOSTATICS2(VARARGIN) Magnetic field around a horseshoe magnet.
Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
sfun        string {sflag2}        Shape function for pressure
hmax        scalar {0.01}          Grid size
iorient     scalar 0/{1,2,3}       Magnet orientation (top, right, bottom, left)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'sfun',     'sflag2';
             'hmax',     0.01;
             'iorient',  0;
             'iplot',    1;
             'tol',      1e-1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' };
 fea.geom.objects = { gobj_circle([0 0],0.05,'C1'), ...
                      gobj_circle([0 0],0.025,'C2'), ...
                      gobj_rectangle(-0.06,0.06,0,0.06,'R1'), ...
                      gobj_rectangle(-0.05,-0.025,0,0.06,'R2'), ...
                      gobj_rectangle(0.025,0.05,0,0.06,'R3'), ...
                      gobj_rectangle(-0.15,0.15,-0.2,0.2,'R4') };
 fea = geom_apply_formula( fea, 'C1-C2-R1' );
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', opt.fid );
 fea.grid = gridrotate( fea.grid, -pi/2*opt.iorient );


% Problem definition.
 fea = addphys( fea, @magnetostatics );
 switch( opt.iorient )
   case 0
     if( ~exist('csg_op') )
       fea.physs.eqn.coef{4,end} = { 0 -1 1 0 };
     else
       fea.physs.eqn.coef{4,end} = { 1 -1 0 0 };
     end
   case 1
     if( ~exist('csg_op') )
       fea.physs.eqn.coef{3,end} = { 0 -1 1 0 };
     else
       fea.physs.eqn.coef{3,end} = { 1 -1 0 0 };
     end
   case 2
     if( ~exist('csg_op') )
       fea.physs.eqn.coef{4,end} = { 0 1 -1 0 };
     else
       fea.physs.eqn.coef{4,end} = { -1 1 0 0 };
     end
```

```
    case 3
      if( ~exist('csg_op') )
        fea.physs.eqn.coef{3,end} = { 0 1 -1 0 };
      else
        fea.physs.eqn.coef{3,end} = { -1 1 0 0 };
      end
  end
  fea.physs.sfun = { opt.sfun };


% Parse and solve problem.
  fea       = parsephys( fea );
  fea       = parseprob( fea );
  fea.sol.u = solvestat( fea, 'fid', opt.fid );   % Call to stationary solver.


% Postprocessing.
  if( opt.iplot>0 )
    figure
    postplot( fea, 'surfexpr', 'Az', ...
              'isoexpr', 'Az', 'isolev', 25, ...
              'arrowexpr', fea.physs.eqn.vars{9,2}, 'arrowcolor', 'w', 'arrowspacing', [45 30] )
    title( 'Magnetic potential (surface, iso), and flux density (arrows) ' )
  end


% Error checking.
  Az   = intsubd( fea.physs.eqn.vars{1,2}, fea );
  Mf   = intsubd( fea.physs.eqn.vars{2,2}, fea );
  gAzb = intbdr(  fea.physs.eqn.vars{5,2}, fea );
  Scb1 = intbdr(  fea.physs.bdr.vars{2,2}, fea, 1 );
  Scb2 = intbdr(  fea.physs.bdr.vars{2,2}, fea, 2 );
  Scb3 = intbdr(  fea.physs.bdr.vars{2,2}, fea, 3 );
  Scb4 = intbdr(  fea.physs.bdr.vars{2,2}, fea, 4 );
  out.err = [ abs(Az+9.637833e-11)/9.637833e-11;
              abs(Mf-0.005108)/0.005108;
              abs(gAzb-1.15859e-8)/1.15859e-8;
              abs(Scb1-0.001326)/0.001326;
              abs(Scb2+0.001865)/0.001865;
              abs(Scb3-0.002403)/0.002403;
              abs(Scb4+0.001865)/0.001865 ];
  out.pass = all( out.err < opt.tol );


  if ( nargout==0 )
    clear fea out
  end
```

11.140 examples/ex_magnetostatics3

EX_MAGNETOSTATICS3 3D Cylindrical magnet example.

[FEA, OUT] =

EX_MAGNETOSTATICS3(VARARGIN) 3D Magnetostatic test example for a cylindrical magnet with no electrical currents.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
sfun        string {sflag2}        Shape function for pressure
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'sfun',     'sflag2';
             'iplot',    1;
             'tol',      1e-1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' 'z' };
 fea.geom.objects = { gobj_cylinder([0 0 -0.03],0.03,0.06,3,'C1') ...
                      gobj_cylinder([0 0 -0.01],0.005,0.02,3,'C2') };

 nref = 1;
 g1 = cylgrid(nref*2,nref*3,nref*12,0.005,0.06,[0 0 -0.03]',3);
 g2 = gridextrude( ringgrid(nref*4,nref*12,0.005,0.03,[0;0],pi/4), nref*12, 0.06 );
 g2.p(3,:) = g2.p(3,:) - 0.03;
 fea.grid  = gridmerge( g1, 1:4, g2, 1:4 );
 fea.grid.s(:) = 1;
 fea.grid.s( selcells( fea, '(z>=(-0.01-sqrt(eps))).*(z<=(0.01+sqrt(eps))).*(sqrt(x^2+y^2)<=(0.005


% Problem definition.
```

```
 fea = addphys( fea, @magnetostatics );
 fea.physs.eqn.coef{4,end} = { 0 1 };
 fea.physs.sfun            = { opt.sfun opt.sfun opt.sfun };


% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'fid', opt.fid );   % Call to stationary solver.


% Postprocessing.
 if( opt.iplot>0 )
   figure
   postplot( fea, 'sliceexpr', fea.dvar{1}, 'arrowexpr', fea.physs.eqn.vars{11,2} )
   title( 'Magnetic potential (slice), flux density (arrow)' )
 end


% Error checking.
 [Vmmin Vmmax]  = minmaxsubd( fea.dvar{1}, fea );
 Mf1 = intsubd( fea.physs.eqn.vars{2,2}, fea, 1 );
 Mf2 = intsubd( fea.physs.eqn.vars{2,2}, fea, 2 );
 out.err = [ abs(Vmmin+0.002158)/0.002158 ;
             abs(Vmmax-0.002158)/0.002158 ;
             abs(Mf1-2.693024e-6)/2.693024e-6 ;
             abs(Mf2-3.084971e-7)/3.084971e-7 ];
 out.pass = all( out.err < opt.tol );


 if ( nargout==0 )
   clear fea out
 end
```

11.141 examples/ex_magnetostatics4

EX_MAGNETOSTATICS4 2D Axisymmetric cylindrical magnet example.

[FEA, OUT] =

EX_MAGNETOSTATICS4(VARARGIN) 2D Axisymmetric magnetostatic test example for a cylindrical magnet with no electrical currents.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
```

```
sfun        string {sflag2}        Shape function for pressure
hmax        scalar {0.0025}        Max grid cell size
iplot       scalar 0/{1}           Plot solution (=1)
                                                                                .
Output      Value/(Size)           Description
--------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'sfun',     'sflag2';
             'iplot',    1;
             'hmax',     0.0025;
             'tol',      1e-2;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'r' 'z' };
 fea.geom.objects = { gobj_rectangle( 0, 0.03,  -0.03, 0.03, 'R1' ) ...
                      gobj_rectangle( 0, 0.005, -0.01, 0.01, 'R2' ) };

 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', opt.fid );


% Problem definition.
 fea = addphys( fea, {@magnetostatics 1} );
 fea.physs.eqn.coef{4,end} = { 1 0 };
 fea.physs.sfun            = { opt.sfun };


% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'fid', opt.fid );   % Call to stationary solver.


% Postprocessing.
 if( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', fea.physs.eqn.vars{1,2}, 'arrowexpr', fea.physs.eqn.vars{9,2} )
   title( 'Magnetic potential (surface), flux density (arrow)' )
 end
```

```
% Error checking.
 [Atmin,Atmax]  = minmaxsubd( fea.physs.eqn.vars{1,2}, fea );
 i_rbdr  = findbdr( fea, ['r>=0.03-',num2str(sqrt(eps))] );
 Scb2 = intbdr( fea.physs.bdr.vars{2,2}, fea, i_rbdr );
 out.err = [ abs(Atmax-2.824e-9)/2.814e-9 ;
             abs(Scb2+4.876998e-4)/4.876998e-4 ];
 out.pass = all( out.err < opt.tol );


 if ( nargout==0 )
   clear fea out
 end
```

11.142 examples/ex_magnetostatics5

EX_MAGNETOSTATICS5 Magnetic field test model.

[FEA, OUT] =

EX_MAGNETOSTATICS5(VARARGIN) Magnetic field test model.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
sfun        string {sflag2}        Shape function for pressure
hmax        scalar {0.1}           Grid size
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'sfun',     'sflag2';
             'hmax',     0.1;
             'iplot',    1;
             'tol',      1e-2;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
```

```
 fid       = opt.fid;


% Geometry and grid generation.
 fea.sdim = { 'x' 'y' };
 fea.grid = rectgrid( round(1/opt.hmax) );
 fea.grid.s( selcells(fea,'y<=(0.5+sqrt(eps))') ) = 2;


% Problem definition.
 fea = addphys( fea, @magnetostatics );
 fea.physs.eqn.coef{3,end} = { 1 0 };
 fea.physs.eqn.coef{4,end} = { 0 1 };
 fea.physs.sfun = { opt.sfun };


% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'icub', 2, 'fid', opt.fid );   % Call to stationary solver.


% Postprocessing.
 if( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', fea.physs.eqn.vars{2,2}, ...
             'isoexpr', fea.physs.eqn.vars{2,2}, 'isolev', 25, ...
             'arrowexpr', fea.physs.eqn.vars{8,2}, 'arrowcolor', 'w', 'arrowspacing', [45 30] )
   title( 'Magnetic field' )
 end


% Error checking.
 Az1  = intsubd( fea.physs.eqn.vars{1,2}, fea, 1 );
 Az2  = intsubd( fea.physs.eqn.vars{1,2}, fea, 2 );
 Mf1  = intsubd( fea.physs.eqn.vars{2,2}, fea, 1 );
 Mf2  = intsubd( fea.physs.eqn.vars{2,2}, fea, 2 );
 Scb1 = intbdr(  fea.physs.bdr.vars{2,2}, fea, 1 );
 Scb2 = intbdr(  fea.physs.bdr.vars{2,2}, fea, 2 );
 Scb3 = intbdr(  fea.physs.bdr.vars{2,2}, fea, 3 );
 Scb4 = intbdr(  fea.physs.bdr.vars{2,2}, fea, 4 );
 out.err = [ abs(Az1+3.178289e-8)/3.178289e-8 ;
             abs(Az2+3.178289e-8)/3.178289e-8 ;
             abs(Mf1-0.389419)/0.389419 ;
             abs(Mf2-0.520161)/0.520161 ;
             abs(Scb1-0.16233)/0.16233 ;
             abs(Scb2-0.836552)/0.836552 ;
             abs(Scb3+0.83767)/0.83767 ;
             abs(Scb4+0.16343)/0.16343 ];
 out.pass = all( out.err < opt.tol );
```

```
  if ( nargout==0 )
    clear fea out
  end
```

11.143 examples/ex_planestress1

EX_PLANESTRESS1 Example for plane stress for hole in plate.

[FEA, OUT] =

EX_PLANESTRESS1(VARARGIN) Example to calculate displacements and stresses for a hole in plate configuration under plane stress assumption.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {210e9}         Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
diam        scalar {0.01}          Diameter of hole
thick       scalar {0.001}         Plate thickness
force       scalar {1000}          Load force
sx_ref      scalar {3e7}           Reference stress in x-direction
hmax        scalar {0.00125}       Max grid cell size
sfun        string {sflag1}        Shape function for displacements
iphys       scalar 0/{1}           Use physics mode to define problem (=1)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
  cOptDef = { ...
    'E',        210e9; ...
    'nu',       0.3; ...
    'diam',     0.01; ...
    'thick',    0.001; ...
    'force',    1000; ...
    'sx_ref',   3e7; ...
    'hmax',     0.00125; ...
```

```
    'sfun',     'sflag1'; ...
    'iphys',    1; ...
    'iplot',    1; ...
    'tol',      0.05; ...
    'fid',      1 };
  [got,opt] = parseopt(cOptDef,varargin{:});
  fid       = opt.fid;


% Model, geometry, and grid parameters.
  h         = 0.05;            % Height of 1/4 rectangular domain.
  l         = 0.05;            % Length of 1/4 rectangular domain.
  xc        = 0;               % x-coordinate of hole center.
  yc        = 0;               % y-coordinate of hole center.
  area      = 2*h*opt.thick;   % Area on which load force is applied.


% Geometry definition.
  gobj1 = gobj_rectangle( 0, l, 0, h, 'R1' );
  gobj2 = gobj_circle( [0 0], opt.diam/2, 'C1' );
  fea.geom.objects = { gobj1 gobj2 };
  fea = geom_apply_formula( fea, 'R1-C1' );
  fea.sdim = { 'x' 'y' };   % Coordinate names.


% Grid generation.
  fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
  n_bdr    = max(fea.grid.b(3,:));        % Number of boundaries.


% Boundary conditions.
  dtol  = opt.diam/10;
  lbdr  = findbdr( fea, ['x<=',num2str(dtol)] );     % Left boundary number.
  rbdr  = findbdr( fea, ['x>=',num2str(l-dtol)] );   % Right boundary number.
  lobdr = findbdr( fea, ['y<=',num2str(dtol)] );     % Lower boundary number.


% Problem definition.
  E11 = opt.E/(1-opt.nu^2);
  E12 = opt.nu*E11;
  E22 = E11;
  E33 = opt.E/(1+opt.nu)/2;

  if ( opt.iphys==1 )

    fea = addphys(fea,@planestress);      % Add plane stress physics mode.
    fea.phys.pss.eqn.coef{1,end} = { opt.nu };
    fea.phys.pss.eqn.coef{2,end} = { opt.E  };
    fea.phys.pss.sfun            = { opt.sfun opt.sfun };   % Set shape functions.

    bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
    bctype{1,lbdr}  = 1;
```

```
    bctype{2,lobdr} = 1;
    fea.phys.pss.bdr.coef{1,5}   = bctype;
    bccoef = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
    bccoef{1,rbdr} = opt.force/area;
    fea.phys.pss.bdr.coef{1,end} = bccoef;

    fea = parsephys(fea);                % Check and parse physics modes.

  else

    fea.dvar  = { 'u' 'v' };                  % Dependent variable names.
    fea.sfun  = { opt.sfun opt.sfun };        % Shape functions.

% Define equation system.
    fea.eqn.a.form = { [2 3;2 3] [3 2;2 3];   ...
                       [3 2;2 3] [2 3;2 3] }; ...
    fea.eqn.a.coef = { {E11 E33} {E12 E33};   ...
                       {E33 E12} {E33 E11} };
    fea.eqn.f.form = { 1 1 };
    fea.eqn.f.coef = { 0 0 };

% Define boundary conditions.
    fea.bdr.d     = cell(2,n_bdr);
    fea.bdr.n     = cell(2,n_bdr);
   [fea.bdr.n{:}] = deal(0);              % Assign zero to all Neumann boundaries.

    fea.bdr.n{1,rbdr}  = opt.force/area;  % Set horizontal load force on right boundary.

    fea.bdr.d{1,lbdr}  = 0;               % Set zero horizontal displacement on left boundary.

    fea.bdr.d{2,lobdr} = 0;               % Set zero vertical displacement on lower boundary.

  end


% Parse and solve problem.
  fea       = parseprob(fea);             % Check and parse problem struct.
  fea.sol.u = solvestat(fea,'fid',fid);   % Call to stationary solver.


% Postprocessing.
  s_sx = [num2str(E11),'*ux+',num2str(E12),'*vy'];
  if ( opt.iplot>0 )
    figure
    postplot(fea,'surfexpr',s_sx,'isoexpr',s_sx,'evaltype','exact')
    title('Stress, x-component')
  end


% Error checking.
  [~,sx_max] = minmaxsubd( s_sx, fea );
  out.sx_max = sx_max;
```

```
  out.err    = opt.sx_ref-sx_max;
  out.pass   = (sx_max>(opt.sx_ref*(1-opt.tol)))&&(sx_max<(opt.sx_ref*(1+opt.tol)));


  if ( nargout==0 )
    clear fea out
  end
```

11.144 examples/ex_planestress2

EX_PLANESTRESS2 NAFEMS benchmark challenge 1 plane stress example.

[FEA, OUT] =

EX_PLANESTRESS2(VARARGIN) NAFEMS benchmark example to calculate von Mieses stress thin plate under plane stress assumption with loads all around.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {210e9}         Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {1/20}          Max grid cell size
sfun        string {sflag1}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
  cOptDef = { ...
    'E',        210e9; ...
    'nu',       0.3; ...
    'igrid',    0; ...
    'hmax',     1/20; ...
    'sfun',     'sflag1'; ...
    'iplot',    1; ...
    'fid',      1 };
```

```
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 gobj1 = gobj_rectangle( 0, 1, 0, 1, 'R1' );
 fea.geom.objects = { gobj1 };
 fea.sdim = { 'x' 'y' };


% Grid generation.
 if( opt.igrid==1 )
   fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 else
   nx = 1/opt.hmax;
   fea.grid = rectgrid( nx, nx );
 end

% Boundary conditions.
 dtol = 0.1;
 if( opt.igrid==1 )
   lbdr = findbdr( fea, ['x<',num2str(dtol)] );     % Left boundary number.
   rbdr = findbdr( fea, ['x>',num2str(1-dtol)] );   % Right boundary number.
   tbdr = findbdr( fea, ['y>',num2str(1-dtol)] );   % Top boundary number.
   bbdr = findbdr( fea, ['y<',num2str(dtol)] );     % Bottom boundary number.
 else
   lbdr = 4;
   rbdr = 2;
   tbdr = 3;
   bbdr = 1;
 end


% Add plane stress physics mode.
 fea = addphys(fea,@planestress);
 fea.phys.pss.eqn.coef{1,end} = { opt.nu };
 fea.phys.pss.eqn.coef{2,end} = { opt.E  };
 fea.phys.pss.sfun            = { opt.sfun opt.sfun };

% Set boundary condition types.
 bctype = mat2cell( zeros(2,4), [1 1], [1 1 1 1] );
 fea.phys.pss.bdr.coef{1,5}   = bctype;

% Set loads on boundary.
 bccoef = mat2cell( zeros(2,4), [1 1], [1 1 1 1] );
 bccoef{1,tbdr} = '-x';
 bccoef{2,tbdr} = 'x';
 bccoef{1,bbdr} = '-(1-x)';
 bccoef{2,bbdr} = '1-x';
 bccoef{1,lbdr} = '1-y';
 bccoef{2,lbdr} = '-(1-y)';
 bccoef{1,rbdr} = 'y';
```

```
 bccoef{2,rbdr} = '-y';
 fea.phys.pss.bdr.coef{1,end} = bccoef;


% Parse and solve problem.
 fea       = parsephys(fea);             % Check and parse physics modes.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid);   % Call to stationary solver.


% Postprocessing.
 s_vm = fea.phys.pss.eqn.vars{1,2};
 if ( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', s_vm, 'isoexpr', s_vm )
   title('von Mieses stress')
 end

% Error checking.
 dtol = sqrt(eps);
 sm  = evalexpr( s_vm, [0.5;0.5], fea );
 s01 = evalexpr( s_vm, [dtol;1],    fea );
 s10 = evalexpr( s_vm, [1;dtol],    fea );
 s00 = evalexpr( s_vm, [dtol;dtol], fea );
 s11 = evalexpr( s_vm, [1;1],       fea );

 sol = [ sm s01 s10 s00 s11 ];
 ref = [  0   0   0   2   2 ];
 out.err    = norm(sol-ref)/norm(ref);
 out.pass   = out.err < 0.1;


 if ( nargout==0 )
   clear fea out
 end
```

11.145 examples/ex_planestress3

EX_PLANESTRESS3 NAFEMS benchmark model for stress of a tapered membrane.

[FEA, OUT] =

EX_PLANESTRESS3(VARARGIN) NAFEMS benchmark for stress of a tapered membrane. Two test cases are modeled, the first with the x displacements fixes on the left boundary while a horizontal force is applied to the right boundary. Secondly a gravitational body force is applied while the left boundary is fixed. Reference: Linear Statics Benchmarks Vol. 1, NAFEMS Ltd., 1987.

Accepts the following property/value pairs.

```
   Input       Value/{Default}        Description
   -------------------------------------------------------------------------------------
   E           scalar {210e9}         Modulus of elasticity
   nu          scalar {0.3}           Poissons ratio
   thick       scalar {0.1}           Plate thickness
   hmax        scalar {1/20}          Max grid cell size
   sfun        string {sflag1}        Shape function for displacements
   iplot       scalar 0/{1}           Plot solution (=1)

   Output      Value/(Size)           Description
   -------------------------------------------------------------------------------------
   fea         struct                 Problem definition struct
   out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'E',        210e9; ...
   'nu',       0.3; ...
   'thick',    0.1; ...
   'icase',    1; ...
   'hmax',     0.3; ...
   'sfun',     'sflag2'; ...
   'iplot',    1; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 gobj1 = gobj_polygon( [0 4 4 0 0;0 1 3 4 0]', 'P1' );
 fea.geom.objects = { gobj1 };
 fea.sdim = { 'x' 'y' };


% Grid generation.
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );


% Add plane stress physics mode.
 fea = addphys(fea,@planestress);
 fea.phys.pss.eqn.coef{1,end} = { opt.nu };
 fea.phys.pss.eqn.coef{2,end} = { opt.E  };
 fea.phys.pss.sfun            = { opt.sfun opt.sfun };
```

```
 if( opt.icase==2 )
   fea.phys.pss.eqn.coef{4,end} = { -9.81*7000 };
 end


% Set boundary conditions.
 dtol = 0.1;
 lbdr = findbdr( fea, ['x<',num2str(dtol)] );     % Left boundary number.
 rbdr = findbdr( fea, ['x>',num2str(1-dtol)] );   % Right boundary number.
 n_bdr  = max(fea.grid.b(3,:));        % Number of boundaries.
 bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bccoef = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 if( opt.icase==1 )
   bctype{1,lbdr} = 1;
   bccoef{1,rbdr} = 1e7/opt.thick;
 else
   bctype{1,lbdr} = 1;
   bctype{2,lbdr} = 1;
 end
 fea.phys.pss.bdr.coef{1,end} = bccoef;
 fea.phys.pss.bdr.coef{1,5}   = bctype;


% Parse and solve problem.
 fea       = parsephys(fea);             % Check and parse physics modes.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid);   % Call to stationary solver.


% Postprocessing.
 if( opt.icase==1 )
   s_title = fea.phys.pss.eqn.vars{5,1};
   s_expr  = fea.phys.pss.eqn.vars{5,2};
 else
   s_title = fea.phys.pss.eqn.vars{7,1};
   s_expr  = fea.phys.pss.eqn.vars{7,2};
 end
 if ( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', s_expr, 'isoexpr', s_expr )
   title( s_title )
 end


% Error checking.
 s_02  = evalexpr( s_expr, [0;2], fea );
 s_ref = (61.3e6*(opt.icase==1)) + (-0.2e6*(opt.icase==2));
 out.stress = s_02;
 out.err    = abs(s_02-s_ref)/s_ref;
 out.pass   = out.err<0.01;
```

```
  if ( nargout==0 )
    clear fea out
  end
```

11.146 examples/ex_planestress4

EX_PLANESTRESS4 Example of thermally induced stress.

[FEA, OUT] =

EX_PLANESTRESS4(VARARGIN) NAFEMS T1 Benchmark example for thermally induced stress. A 20x20 mm plate is subjected to thermal load at a circular spot with radius 1 mm. Plane stress and symmetry can be assumed. The stress in the y-direction is sought just outside the thermal spot.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {100e9}         Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
hmax        scalar {4e-4}          Max grid cell size
sfun        string {sflag1}        Shape function for displacements
iphys       scalar 0/{1}           Use physics mode to define problem (=1)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
  cOptDef = { ...
    'E',        100e9; ...
    'nu',       0.3; ...
    'sy_ref',   50e6; ...
    'hmax',     4e-4; ...
    'sfun',     'sflag2'; ...
    'igeom',    1; ...
    'iphys',    1; ...
    'iplot',    1; ...
```

```
    'tol',      0.1; ...
    'fid',      1 };
  [got,opt] = parseopt( cOptDef, varargin{:} );
  fid       = opt.fid;


% Model, geometry, and grid parameters.
  l  = 1e-2;   % Length of quarter domain.
  r  = 1e-3;   % Radius of thermal spot.
  xc = 0;      % x-coordinate of spot center.
  yc = 0;      % y-coordinate of spot center.


% Geometry definition.
  switch( opt.igeom )
    case 1
      gobj1 = gobj_rectangle( 0, l, 0, l, 'R1' );
      gobj2 = gobj_circle( [0 0], r, 'C1' );
      gobj3 = gobj_rectangle( 0, l, 0, l, 'R2' );
      gobj4 = gobj_circle( [0 0], r, 'C2' );
      fea.geom.objects = { gobj1 gobj2 gobj3 gobj4 };
      fea.geom = geom_apply_formula( fea.geom, 'R1-C1' );
      fea.geom = geom_apply_formula( fea.geom, 'R2&C2' );
    case 2
      gobj1 = gobj_rectangle( 0, l, 0, l, 'R1' );
      gobj2 = gobj_circle( [0 0], r, 'C1' );
      gobj3 = gobj_rectangle( 0, l, 0, l, 'R2' );
      fea.geom.objects = { gobj1 gobj2 gobj3 };
      fea.geom = geom_apply_formula( fea.geom, 'R1&C1' );
    case 3
      gobj1 = gobj_rectangle( 0, l, 0, l );
      gobj2 = gobj_circle(  [0 0], r );
      gobj3 = gobj_polygon( [0 0; r 0;r -r; -r -r; -r r; 0 r] );
      fea.geom.objects = { gobj1 gobj2 gobj3 };
      fea.geom = geom_apply_formula( fea.geom, 'C1-P1' );
  end
  fea.sdim = { 'x' 'y' };   % Coordinate names.


% Grid generation.
  fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );
  n_bdr    = max(fea.grid.b(3,:));   % Number of boundaries.


% Boundary conditions.
  dtol  = r/5;
  lbdr  = findbdr( fea, ['x<=',num2str(dtol)] );   % Left boundary number.
  lobdr = findbdr( fea, ['y<=',num2str(dtol)] );   % Lower boundary number.


% Problem definition.
  if( opt.iphys==1 )
```

```
    fea = addphys( fea, @planestress );
    fea.phys.pss.eqn.coef{1,end} = { opt.nu };
    fea.phys.pss.eqn.coef{2,end} = { opt.E  };
    if( opt.igeom==1 )
      fea.phys.pss.eqn.coef{5,end} = { 0, 1e-3 };
    else
      fea.phys.pss.eqn.coef{5,end} = { 1e-3, 0 };
    end
    fea.phys.pss.eqn.coef{6,end} = { 1 };
    fea.phys.pss.sfun            = { opt.sfun opt.sfun };

    bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
    bctype{1,lbdr(1)}  = 1;
    bctype{1,lbdr(2)}  = 1;
    bctype{2,lobdr(1)} = 1;
    bctype{2,lobdr(2)} = 1;
    fea.phys.pss.bdr.coef{1,5} = bctype;

    fea = parsephys(fea);

    s_sy = fea.phys.pss.eqn.vars{6,end};
  else

    E11 = opt.E/(1-opt.nu^2);
    E12 = opt.nu*E11;
    E22 = E11;
    E33 = opt.E/(1+opt.nu)/2;
    if( opt.igeom==1 )
      fea.expr = { 'alfaT', { 0 opt.E/(1-opt.nu)*1e-3 } };
    else
      fea.expr = { 'alfaT', { opt.E/(1-opt.nu)*1e-3 0 } };
    end

    fea.dvar  = { 'u' 'v' };
    fea.sfun  = { opt.sfun opt.sfun };

% Define equation system.
    fea.eqn.a.form = { [2 3;2 3] [3 2;2 3];   ...
                       [3 2;2 3] [2 3;2 3] }; ...
    fea.eqn.a.coef = { {E11 E33} {E12 E33};   ...
                       {E33 E12} {E33 E11} };
    fea.eqn.f.form = { 2 3 };
    fea.eqn.f.coef = { 'alfaT' 'alfaT' };

% Define boundary conditions.
    fea.bdr.d     = cell(2,n_bdr);
    fea.bdr.n     = cell(2,n_bdr);
   [fea.bdr.n{:}] = deal(0);

    fea.bdr.d{1,lbdr(1)}  = 0;
    fea.bdr.d{1,lbdr(2)}  = 0;
```

```
    fea.bdr.d{2,lobdr(1)} = 0;
    fea.bdr.d{2,lobdr(2)} = 0;

    s_sy = [num2str(E12),'*ux + ',num2str(E11),'*vy - alfaT'];
  end

% Parse and solve problem.
  fea       = parseprob( fea );               % Check and parse problem struct.
  fea.sol.u = solvestat( fea, 'fid', fid );   % Call to stationary solver.


% Postprocessing.
  if( opt.iplot>0 )
    figure
    postplot( fea, 'surfexpr', s_sy, 'isoexpr', s_sy, 'isolev', 50 )
    title( 'Stress, y-component' )
  end


% Error checking.
  sy_D     = evalexpr( s_sy, [1e-3+sqrt(eps);0], fea );
  out.sy_D = sy_D;
  out.err  = abs(sy_D - opt.sy_ref)/opt.sy_ref;
  out.pass = out.err <= opt.tol;


  if( nargout==0 )
    clear fea out
  end
```

11.147 examples/ex_planestress5

EX_PLANESTRESS5 Plane stress example for an elliptic membrane.

[FEA, OUT] =

EX_PLANESTRESS5(VARARGIN) Example to calculate displacements and stresses for an elliptic membrane with a hole in it. NAFEMS benchmark example LE1.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {210e9}         Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
hmax        scalar {0.1}           Max grid cell size
sfun        string {sflag2}        Shape function for displacements
```

```
iplot       scalar 0/{1}           Plot solution (=1)
                                                                                .
Output      Value/(Size)           Description
--------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'E',        210e9; ...
   'nu',       0.3; ...
   'hmax',     0.1; ...
   'sfun',     'sflag2'; ...
   'iplot',    1; ...
   'tol',      0.05; ...
   'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;


% Geometry definition.
 gobj1 = gobj_ellipse( [0 0], 3.25, 2.75, 'E1' );
 gobj2 = gobj_ellipse( [0 0], 2, 1, 'E2' );
 gobj3 = gobj_rectangle( -3.25, 3.25, -2.75, 0, 'R1' );
 gobj4 = gobj_rectangle( -3.25, 0, 0, 2.75, 'R2' );
 fea.geom.objects = { gobj1 gobj2 gobj3 gobj4 };
 fea = geom_apply_formula( fea, 'E1-E2-R1-R2' );
 fea.sdim = { 'x' 'y' };


% Grid generation.
 fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );
 n_bdr    = max(fea.grid.b(3,:));   % Number of boundaries.


% Boundary conditions.
 dtol  = sqrt(eps);
 lbdr  = findbdr( fea, ['x<=',num2str(dtol)] );   % Left boundary number.
 lobdr = findbdr( fea, ['y<=',num2str(dtol)] );   % Lower boundary number.


% Problem definition.
 E11 = opt.E/(1-opt.nu^2);
 E12 = opt.nu*E11;
 E22 = E11;
 E33 = opt.E/(1+opt.nu)/2;
```

```
 fea = addphys(fea,@planestress);      % Add plane stress physics mode.
 fea.phys.pss.eqn.coef{1,end} = { opt.nu };
 fea.phys.pss.eqn.coef{2,end} = { opt.E  };
 fea.phys.pss.sfun            = { opt.sfun opt.sfun };   % Set shape functions.

 bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bctype{1,lbdr}  = 1;
 bctype{2,lobdr} = 1;
 fea.phys.pss.bdr.coef{1,5} = bctype;

% Add normal load to boundary 1.
 bccoef = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bccoef{1,1} = 'nx*10e6';
 bccoef{2,1} = 'ny*10e6';
 fea.phys.pss.bdr.coef{1,end} = bccoef;


% Parse and solve problem.
 fea       = parsephys(fea);
 fea       = parseprob(fea);
 fea.sol.u = solvestat( fea, 'fid', fid );


% Postprocessing.
 s_sy = [num2str(E12),'*ux+',num2str(E11),'*vy'];
 if ( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', s_sy, 'isoexpr', s_sy )
   title('Stress, x-component')
 end


% Error checking.
 sy_D     = evalexpr( s_sy, [2;0]+sqrt(eps)*1e1, fea );
 out.sy_D = sy_D;
 out.err  = abs(sy_D - 92.7e6)/92.7e6;
 out.pass = out.err <= opt.tol;


 if ( nargout==0 )
   clear fea out
 end
```

11.148 examples/ex_planestress6

EX_PLANESTRESS6 Plane stress analysis of a pressure vessel.

[FEA, OUT] =

EX_PLANESTRESS6(VARARGIN) Model example for plain stress approximation of a pressure vessel (annular cross section with symmetry).

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {207e9}         Modulus of elasticity
nu          scalar {0.27}          Poissons ratio
sfun        string {sflag1}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'E',        207e9; ...
   'nu',       0.27; ...
   'sfun',     'sflag1'; ...
   'iplot',    1; ...
   'igrid',    1; ...
   'tol',      0.1; ...
   'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;


% Geometry and grid.
 fea.sdim = { 'x' 'y' };   % Coordinate names.
 fea.grid = ringgrid( 12, 216, 100e-3, 120e-3 );
 fea.grid = delcells( fea.grid, selcells( fea.grid, '(x<=eps) + (y<=eps)') );
 if( opt.igrid~=1 )
   fea.grid = quad2tri( fea.grid );
 end
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Problem definition.
 fea = addphys( fea, @planestress );
 fea.phys.pss.eqn.coef{1,end} = { opt.nu };
 fea.phys.pss.eqn.coef{2,end} = { opt.E  };
```

```
 fea.phys.pss.sfun            = { opt.sfun opt.sfun };


% Boundary conditions.
 bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bctype{1,4} = 1;
 bctype{2,3} = 1;
 fea.phys.pss.bdr.coef{1,5} = bctype;

 bccoef = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bccoef{1,1} = '-nx*1e4';
 bccoef{2,1} = '-ny*1e4';
 fea.phys.pss.bdr.coef{1,end} = bccoef;


% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );                          % Check and parse problem struct.
 fea.sol.u = solvestat( fea, 'fid', fid, 'icub', 1+str2num(strrep(opt.sfun,'sflag','')) );
% Call to stationary solver.


% Postprocessing.
 s_disp = fea.phys.pss.eqn.vars{2,end};
 if( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', s_disp )
   title( 'Total displacement' )
 end


% Error checking.
 s_sx     = fea.phys.pss.eqn.vars{5,end};
 s_sy     = fea.phys.pss.eqn.vars{6,end};
 s_sxy    = fea.phys.pss.eqn.vars{7,end};
 s_sp1    = fea.phys.pss.eqn.vars{8,end};
 s_sp3    = fea.phys.pss.eqn.vars{10,end};
 s_ez     = fea.phys.pss.eqn.vars{13,end};
 s_ep1    = fea.phys.pss.eqn.vars{15,end};
 s_ep2    = fea.phys.pss.eqn.vars{16,end};
 s_ep3    = fea.phys.pss.eqn.vars{17,end};
 v_disp   = evalexpr( s_disp, [100e-3 120e-3-2*sqrt(eps);0 0]+sqrt(eps), fea )';
 v_dref   = [2.809e-8 2.635e-8];
 [v_sx(1),v_sx(2)] = minmaxsubd( s_sx, fea );
 v_sxref  = [-10000 55454];
 [v_sy(1),v_sy(2)] = minmaxsubd( s_sy, fea );
 v_syref  = [-10000 55454];
 [v_sxy(1),v_sxy(2)] = minmaxsubd( s_sxy, fea );
 v_sxyref = [-32730 0];
 [v_sp1(1),v_sp1(2)] = minmaxsubd( s_sp1, fea );
 v_sp1ref = [4.5e4 55454];
 [v_sp3(1),v_sp3(2)] = minmaxsubd( s_sp3, fea );
```

```
  v_sp3ref = [-1e4 0];
  [v_ez(1),v_ez(2)] = minmaxsubd( s_ez, fea );
  v_ezref = [-5.929e-8 -5.929e-8];
  [v_ep1(1),v_ep1(2)] = minmaxsubd( s_ep1, fea );
  v_ep1ref = [2.196e-7 2.809e-7];
  [v_ep2(1),v_ep2(2)] = minmaxsubd( s_ep2, fea );
  v_ep2ref = [-5.929e-8 -5.929e-8];
  [v_ep3(1),v_ep3(2)] = minmaxsubd( s_ep3, fea );
  v_ep3ref = [-1.206e-7 -5.929e-8];
  out.err  = [ abs([v_dref-v_disp])./v_dref ;
               abs([v_sxref-v_sx])./v_sxref ;
               abs([v_syref-v_sy])./v_syref ;
               abs([v_sxyref(1)-v_sxy(1)])./v_sxyref(1) 0 ;
               abs([v_sp1ref(2)-v_sp1(2)])./v_sp1ref(2) 0 ;
               abs([v_sp3ref(1)-v_sp3(1)])./v_sp3ref(1) 0 ;
               abs([v_ezref(1)-v_ez(1)])./v_ezref(1) 0 ;
               abs([v_ep1ref(1)-v_ep1(1)])./v_ep1ref(1) 0 ;
               abs([v_ep2ref(1)-v_ep2(1)])./v_ep2ref(1) 0 ;
               abs([v_ep3ref(1)-v_ep3(1)])./v_ep3ref(1) 0 ];
  out.pass = all( out.err(:) <= opt.tol );


  if( nargout==0 )
    clear fea out
  end
```

11.149 examples/ex_planestrain1

EX_PLANESTRAIN1 Plane strain analysis of a pressure vessel.

[FEA, OUT] =

EX_PLANESTRAIN1(VARARGIN) Benchmark example for plain strain approximation of a pressure vessel (annular cross section with symmetry).

Reference. B. J. Mac Donald, Practical Stress Analysis with Finite Elements (2nd Ed), case study E on page 327, 2007.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
E           scalar {207e9}         Modulus of elasticity
nu          scalar {0.27}          Poissons ratio
sfun        string {sflag1}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
```

```
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { ...
   'E',        207e9; ...
   'nu',       0.27; ...
   'sfun',     'sflag1'; ...
   'iplot',    1; ...
   'igrid',    1; ...
   'tol',      0.1; ...
   'fid',      1 };
 [got,opt] = parseopt( cOptDef, varargin{:} );
 fid       = opt.fid;


% Geometry and grid.
 fea.sdim = { 'x' 'y' };   % Coordinate names.
 fea.grid = ringgrid( 12, 216, 100e-3, 120e-3 );
 fea.grid = delcells( fea.grid, selcells( fea.grid, '(x<=eps) + (y<=eps)') );
 if( opt.igrid~=1 )
   fea.grid = quad2tri( fea.grid );
 end
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Problem definition.
 fea = addphys( fea, @planestrain );
 fea.phys.psn.eqn.coef{1,end} = { opt.nu };
 fea.phys.psn.eqn.coef{2,end} = { opt.E  };
 fea.phys.psn.sfun            = { opt.sfun opt.sfun };


% Boundary conditions.
 bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bctype{1,4} = 1;
 bctype{2,3} = 1;
 fea.phys.psn.bdr.coef{1,5} = bctype;

 bccoef = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bccoef{1,1} = '-nx*1e4';
 bccoef{2,1} = '-ny*1e4';
 fea.phys.psn.bdr.coef{1,end} = bccoef;
```

```
% Parse and solve problem.
 fea       = parsephys( fea );
 fea       = parseprob( fea );                          % Check and parse problem struct.
 fea.sol.u = solvestat( fea, 'fid', fid, 'icub', 1+str2num(strrep(opt.sfun,'sflag','')) );
% Call to stationary solver.


% Postprocessing.
 s_disp = fea.phys.psn.eqn.vars{2,end};
 if( opt.iplot>0 )
   figure
   postplot( fea, 'surfexpr', s_disp )
   title( 'Total displacement' )
 end


% Error checking.
 s_sx     = fea.phys.psn.eqn.vars{5,end};
 s_sy     = fea.phys.psn.eqn.vars{6,end};
 s_sxy    = fea.phys.psn.eqn.vars{8,end};
 s_sp1    = fea.phys.psn.eqn.vars{9,end};
 s_sp2    = fea.phys.psn.eqn.vars{10,end};
 s_sp3    = fea.phys.psn.eqn.vars{11,end};
 v_disp   = evalexpr( s_disp, [100e-3 120e-3-2*sqrt(eps);0 0]+sqrt(eps), fea )';
 v_dref   = [2.64e-8 2.41e-8];
 [v_sx(1),v_sx(2)] = minmaxsubd( s_sx, fea );
 v_sxref  = [-10000 55454];
 [v_sy(1),v_sy(2)] = minmaxsubd( s_sy, fea );
 v_syref  = [-10000 55454];
 [v_sxy(1),v_sxy(2)] = minmaxsubd( s_sxy, fea );
 v_sxyref = [-32730 0];
 [v_sp1(1),v_sp1(2)] = minmaxsubd( s_sp1, fea );
 v_sp1ref = [4.5e4 55454];
 [v_sp2(1),v_sp2(2)] = minmaxsubd( s_sp2, fea );
 v_sp2ref = [1.227e4 1.227e4];
 [v_sp3(1),v_sp3(2)] = minmaxsubd( s_sp3, fea );
 v_sp3ref = [-1e4 0];
 out.err  = [ abs([v_dref-v_disp])./v_dref ;
              abs([v_sxref-v_sx])./v_sxref ;
              abs([v_syref-v_sy])./v_syref ;
              abs([v_sxyref(1)-v_sxy(1)])./v_sxyref(1) 0 ;
              abs([v_sp1ref(2)-v_sp1(2)])./v_sp1ref(2) 0 ;
              abs([v_sp2ref-v_sp2])./v_sp2ref      ;
              abs([v_sp3ref(1)-v_sp3(1)])./v_sp3ref(1) 0 ];
 out.pass = all( out.err(:) <= opt.tol );


 if( nargout==0 )
   clear fea out
 end
```

11.150 examples/ex_axistressstrain1

EX_AXISTRESSSTRAIN1 Example for hollow cylider axisymmetric stress-strain.

[FEA, OUT] =

EX_AXISTRESSSTRAIN1(VARARGIN) Example to calculate displacements and stresses in a hollow cylinder in axisymmetric/cylindrical coordinates.

Ref. 4.1.9 Long (generalized plane strain) cylinder subjected to internal and external pressure.

[1] Applied Mechanics of Solids, Allan F. Bower, 2012 (http://solidmechanics.org/).

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
a           scalar {1}             Cylinder inner radius
b           scalar {2}             Cylinder outer radius
p           scalar {20e4}          Load force
E           scalar {200e9}         Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
igrid       scalar 0/{1}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.1}           Max grid cell size
sfun        string {sflag2}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'a',        1;
             'b',        2;
             'p',        20e4;
             'E',        200e9;
             'nu',       0.3;
             'igrid',    0;
             'hmax',     0.1;
             'sfun',     'sflag2';
             'iplot',    1;
             'tol',      1e-3;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
```

```
 fid       = opt.fid;


% Geometry and grid.
 a = opt.a;
 b = opt.b;
 fea.geom.objects = { gobj_rectangle( a, b, 0, 1, 'R1' ) };
 if ( opt.igrid==1 )
   fea.grid = gridgen( fea, 'hmax', opt.hmax, 'fid', fid );
 else
   fea.grid = rectgrid( ceil(1/opt.hmax), ceil(1/opt.hmax), [a b;0 1] );
   if( opt.igrid<0 )
     fea.grid = quad2tri( fea.grid );
   end
 end
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Axisymmetric stress-strain equation definitions.
 fea.sdim = { 'r', 'z' };
 fea = addphys( fea, @axistressstrain );
 fea.phys.css.eqn.coef{1,end} = { opt.nu };
 fea.phys.css.eqn.coef{2,end} = { opt.E  };
 fea.phys.css.sfun            = { opt.sfun opt.sfun };   % Set shape functions.

% Boundary conditions.
 bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 [bctype{2,:}] = deal( 1 );
 fea.phys.css.bdr.coef{1,5} = bctype;

 bccoef = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bccoef{1,4} = opt.p;
 fea.phys.css.bdr.coef{1,end} = bccoef;


% Solve.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'icub', 1+str2num(strrep(opt.sfun,'sflag','')), 'fid', fid );


% Postprocessing.
 n = 20;
 r = linspace(a,b,n);
 z = 0.5*ones(1,n);
 u_ref = opt.p * a^2*(1+opt.nu)*(b^2+r'.^2*(1-2*opt.nu)) ./ (opt.E*(b^2-a^2)*r');
% From Ex 4.19 http://solidmechanics.org/text/Chapter4_1/Chapter4_1.htm
 u = evalexpr( 'r*u', [r;z], fea );
 if( opt.iplot>0 )
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'r*u' )
   title('r-displacement')
```

```
    subplot(1,2,2), hold on
    plot(u_ref,r,'r-')
    plot(u,r,'b.')
    legend('exact solution','computed solution')
    xlabel('r')
    grid on
  end


% Error checking.
  out.err  = norm( u_ref - u )/norm( u_ref );
  out.pass = out.err < opt.tol;


  if( nargout==0 )
    clear fea out
  end
```

11.151 examples/ex_axistressstrain2

EX_AXISTRESSSTRAIN2 Example for a pressurized hollow sphere axisymmetric stress-strain.

[FEA, OUT] =

EX_AXISTRESSSTRAIN2(VARARGIN) Example to calculate displacements and stresses in a pressurized hollow sphere in axisymmetric/cylindrical coordinates.

Ref. 4.1.4 Pressurized hollow sphere.

[1] Applied Mechanics of Solids, Allan F. Bower, 2012 (http://solidmechanics.org/).

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
a           scalar {1}             Cylinder inner radius
b           scalar {2}             Cylinder outer radius
p           scalar {20e4}          Load force
E           scalar {200e9}         Modulus of elasticity
nu          scalar {0.3}           Poissons ratio
igrid       scalar 0/{<0}          Cell type (0=quadrilaterals, <0=triangles)
sfun        string {sflag2}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'a',        1;
             'b',        2;
             'p',        20e4;
             'E',        200e9;
             'nu',       0.3;
             'igrid',    0;
             'sfun',     'sflag2';
             'iplot',    1;
             'tol',      5e-3;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid.
 a = opt.a;
 b = opt.b;
 if ( opt.igrid==1 )
   error('ex_axistressstrain2: unstructured grid not supported.')
 else
   fea.grid = ringgrid( 12, 72, a, b );
   fea.grid = delcells( fea.grid, selcells( fea.grid, '(x<=eps) + (y<=eps)') );
   if( opt.igrid<0 )
     fea.grid = quad2tri( fea.grid );
   end
 end
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Axisymmetric stress-strain equation definitions.
 fea.sdim = { 'r', 'z' };
 fea = addphys( fea, @axistressstrain );
 fea.phys.css.eqn.coef{1,end} = { opt.nu };
 fea.phys.css.eqn.coef{2,end} = { opt.E  };
 fea.phys.css.sfun            = { opt.sfun opt.sfun };   % Set shape functions.


% Boundary conditions.
 bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bctype{1,4} = 1;
 bctype{2,3} = 1;
 fea.phys.css.bdr.coef{1,5} = bctype;

 bccoef = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bccoef{1,1} = ['-r*nr*',num2str(opt.p),];
 bccoef{2,1} = ['-r*nz*',num2str(opt.p),];
 fea.phys.css.bdr.coef{1,end} = bccoef;
```

```
% Solve.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'icub', 1+str2num(strrep(opt.sfun,'sflag','')), 'fid', fid );


% Postprocessing.
 n = 20;
 r = linspace(a,b,n);
 z = zeros(1,n);
 u_ref = 1./(2*opt.E*(b^3-a^3)*r'.^2) .* (2*(opt.p*a^3)*(1-2*opt.nu)*r'.^3+opt.p*(1+opt.nu)*b^3*a^
 u = evalexpr( 'r*u', [r;z], fea );
 if( opt.iplot>0 )
   subplot(1,2,1)
   postplot( fea, 'surfexpr', 'sqrt((r*u)^2+w^2)', 'arrowexpr', {'r*u' 'w'} )
   title('computed displacement')
   subplot(1,2,2), hold on
   plot(u_ref,r,'r-')
   plot(u,r,'b.')
   title( 'radial displacement')
   legend('exact solution','computed solution')
   xlabel('r')
   grid on
 end


% Error checking.
 out.err  = norm( u_ref - u )/norm( u_ref );
 out.pass = out.err < opt.tol;


 if( nargout==0 )
   clear fea out
 end
```

11.152 examples/ex_axistressstrain3

EX_AXISTRESSSTRAIN3 Disc with fixed edge and central point load axisymmetric stress-strain.

[FEA, OUT] =

EX_AXISTRESSSTRAIN3(VARARGIN) Example to calculate displacements and stresses in a disc with fixed outer edge with a central point load in axisymmetric/cylindrical coordinates.

Ref. Chapter 7. Benchmark 3: Point Loaded SS Circular Plate Bending.

[1] Advanced Finite Element Methods (ASEN 6367) - Spring 2013. Department of Aerospace En-

gineering Sciences University of Colorado at Boulder.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
p           scalar {1e3}           Load force
E           scalar {1e3}           Modulus of elasticity
nu          scalar {1/3}           Poissons ratio
sfun        string {sflag1}        Shape function for displacements
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'p',        1e3;
             'E',        1e3;
             'nu',       1/3;
             'sfun',     'sflag1';
             'iplot',    1;
             'igrid',    1;
             'tol',      0.02;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry and grid.
 a = 0;
 b = 10;
 fea.grid = rectgrid( 20, 20, [a b;-0.5 0.5] );
 if( opt.igrid<0 )
   fea.grid = quad2tri( fea.grid );
 end
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Axisymmetric stress-strain equation definitions.
 fea.sdim = { 'r', 'z' };
 fea = addphys( fea, @axistressstrain );
 fea.phys.css.eqn.coef{1,end} = { opt.nu };
 fea.phys.css.eqn.coef{2,end} = { opt.E  };
 fea.phys.css.sfun            = { opt.sfun opt.sfun };   % Set shape functions.
```

```
% Boundary conditions.
 bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 fea.phys.css.bdr.coef{1,5} = bctype;

 bccoef = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bccoef{2,4} = -opt.p/(2*pi);
 fea.phys.css.bdr.coef{1,end} = bccoef;


% Set point constraint w=0 at (b,0).
 [~,ix] = min( (fea.grid.p(1,:)-b).^2 + (fea.grid.p(2,:)-0).^2 );
 fea.pnt(1).index = ix;
 fea.pnt(1).type  = 'constr';
 fea.pnt(1).dvar  = 2;
 fea.pnt(1).expr  = 0;


% Solve.
 fea       = parsephys( fea );
 fea       = parseprob( fea );
 fea.sol.u = solvestat( fea, 'icub', 1+str2num(strrep(opt.sfun,'sflag','')), 'fid', fid );


% Postprocessing.
 D = opt.E*1^3/(12*(1-opt.nu^2));
 u_ref_r = [num2str(-opt.p),'/(8*pi*',num2str(D),')*(',num2str((3+opt.nu)/(1+opt.nu)-1),'-2*log(r/
 u_ref_z = [num2str(-opt.p),'/(16*pi*',num2str(D),')*(',num2str((3+opt.nu)/(1+opt.nu)),'*(',num2st
 if( opt.iplot>0 )
   subplot(2,2,1)
   postplot( fea, 'surfexpr', 'r*u' )
   title('computed r-displacement')

   subplot(2,2,2)
   postplot( fea, 'surfexpr', u_ref_r )
   title('exact r-displacement')

   subplot(2,2,3)
   postplot( fea, 'surfexpr', 'w' )
   title('computed z-displacement')

   subplot(2,2,4)
   postplot( fea, 'surfexpr', u_ref_z )
   title('exact z-displacement')
 end


% Error checking.
 u_r = evalexpr( 'r*u', fea.grid.p, fea );
 u_z = evalexpr( 'w', fea.grid.p, fea );
 u_ref_r = evalexpr( u_ref_r, fea.grid.p, fea );
```

```
  u_ref_z = evalexpr( u_ref_z, fea.grid.p, fea );
  ix = find( ~isnan(u_ref_r + u_ref_z) );
  out.err(1) = norm( u_ref_r(ix) - u_r(ix) )/norm( u_ref_r(ix) );
  out.err(2) = norm( u_ref_z(ix) - u_z(ix) )/norm( u_ref_z(ix) );
  out.pass = all( out.err < opt.tol );


  if( nargout==0 )
    clear fea out
  end
```

11.153 examples/ex_axistressstrain4

EX_AXISTRESSSTRAIN4 Axisymmetric heat induced stress for a brake disk.

[FEA, OUT] =

EX_AXISTRESSSTRAIN4(VARARGIN) Quasi-static axisymmetric simulation of a brake disk under braking. The braking process induces heat through fricition with the brake pad, which in turn results in stresses in the brake disk.

Ref. A. Adamowicz, Axisymmetric FE Model to Analysis of Thermal Stresses in a Brake Disk, Journal of Theoretical and Applied Mechanics, 53, 2, pp. 357-370, Warsaw 2015.

Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
deltad      scalar {5.5e-3}        1/2 Disk thickness
rd          scalar {66e-3}         Disk inner radius
rp          scalar {75.5e-3}       Pad inner radius
Rd          scalar {113.5e-3}      Disk outer radius
sfun        string {sflag2}        Shape function for displacements
nlev        scalar {2}             Uniform grid refinement level
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'deltad',   5.5e-3;
             'rd',       66e-3;
             'rp',       75.5e-3;
             'Rd',       113.5e-3;
             'sfun',     'sflag2';
             'nlev',     2;
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Brake disk material parameters.
 kd     = 1.44e-5;       % Thermal diffsivity [m2/s]
 Kd     = 51;            % Thermal conductivity [W/m/K]
 rhod   = 7100;          % Density [kg/m3]
 cpd    = Kd/kd/rhod;    % Specific heat [J/Kg/K]
 E      = 99.97e9;       % Modulus of elasticity [Pa]
 nu     = 0.29;          % Poisson's ratio
 alpha  = 1.08e-5;       % Coefficient of thermal expansion [1/K]

 T0     = 20 + 273.15;   % Initial temperature [K]

% Brake pad material parameters.
 kp     = 1.46e-5;       % Thermal diffsivity [m2/s]
 Kp     = 34.3;          % Thermal conductivity [W/m/K]
 rhop   = 4700;          % Density [kg/m3]
 cpp    = Kp/kp/rhop;    % Specific heat [J/Kg/K]

% Simulation parameters.
 tmax   = 3.96;          % Maximum time.
 nsteps = 200;           % Number of time steps.


% Create computational geometry and grid.
 fea.sdim = { 'r' 'z' };
 fea.grid = rectgrid( 2^(opt.nlev-1)*45, 2^(opt.nlev-1)*5, [opt.rd opt.Rd;0 opt.deltad] );
 [~,ix] = findbdr( fea, ['(r>=',num2str(opt.rp),') & (z>=',num2str(opt.deltad-sqrt(eps)),')'], 0 )
 fea.grid.b(3,ix) = max(fea.grid.b(3,:)) + 1;   % Add boundary for brake pad.
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Equations and problem definition.
 fea = addphys( fea, @axistressstrain );
 fea.phys.css.eqn.coef{1,end} = { nu    };
 fea.phys.css.eqn.coef{2,end} = { E     };
 fea.phys.css.eqn.coef{5,end} = { alpha };
 fea.phys.css.eqn.coef{6,end} = { ['T-',num2str(T0)] };
 fea.phys.css.sfun            = { opt.sfun opt.sfun };

 fea = addphys( fea, {@heattransfer, 1} );
```

```
 fea.phys.ht.eqn.coef{1,end}  = { rhod };
 fea.phys.ht.eqn.coef{2,end}  = { cpd  };
 fea.phys.ht.eqn.coef{3,end}  = { Kd   };
 fea.phys.ht.sfun             = { opt.sfun };


% Equation coefficients, constants, and expressions.
 fea.expr = { 'eta'    [] [] {} ;
              'f'      [] [] {0.5} ;
              'omega'  [] [] {[]} ;
              'p0'     [] [] {1.47e6} };


% Boundary conditions.
 bctype = mat2cell( zeros(2,n_bdr), [1 1], ones(1,n_bdr) );
 bctype{2,1} = 1;
 fea.phys.css.bdr.coef{1,5} = bctype;

 eta   = sqrt(Kd*rhod*cpd) / (sqrt(Kd*rhod*cpd) + sqrt(Kp*rhop*cpp));
 f     = 0.5;
 p0    = 1.47e6;
 qd    = [num2str(eta*f*88.464*p0/(2*pi-0.8)),'*r*(1-t/',num2str(tmax),')'];

 fea.phys.ht.bdr.sel(5) = 4;
 fea.phys.ht.bdr.coef{4,end}{5}{1} = qd;


% Solve problem.
 fea = parsephys( fea );
 fea = parseprob( fea );
 [fea.sol.u,tlist] = solvetime( fea, 'ischeme', 2, 'icub', 3, 'init', { 0 0 T0 }, 'tmax', tmax, 't


% Postprocessing.
 sr  = fea.phys.css.eqn.vars{5,end};
 st  = fea.phys.css.eqn.vars{6,end};
 svm = fea.phys.css.eqn.vars{1,end};
 if( opt.iplot>0 )
   figure
   solnum = numel(tlist);
   niso   = 25;
   subplot(2,2,1)
   postplot( fea, 'surfexpr', 'T-273.15', 'isoexpr', 'T-273.15', 'isolev', niso, 'solnum', solnum
   title(['Temperature [deg C], t = ',num2str(tlist(solnum)),' s'])
   subplot(2,2,2)
   postplot( fea, 'surfexpr', ['(',sr,')*1e-6'],  'isoexpr', ['(',sr,')*1e-6'],  'isolev', niso, '
   title('radial stress [MPa]')
   subplot(2,2,3)
   postplot( fea, 'surfexpr', ['(',st,')*1e-6'],  'isoexpr', ['(',st,')*1e-6'],  'isolev', niso, '
   title('tangential stress [MPa]')
   subplot(2,2,4)
   postplot( fea, 'surfexpr', ['(',svm,')*1e-6'], 'isoexpr', ['(',svm,')*1e-6'], 'isolev', niso, '
```

```
title('von Mieses stress [MPa]')

figure
u_stored = fea.sol.u;
np = 100;
times = [0.1 0.2 1 2 3 tmax];
rz = [ linspace(opt.rd,opt.Rd,np); opt.deltad*ones(1,np) ];
for i=1:numel(times)
  t_i = times(i);
  ix  = max( find( tlist<t_i ) );
  s   = ( tlist(ix+1) - t_i )/( tlist(ix+1) - tlist(ix) );
  fea.sol.u = u_stored(:,ix) + s*( u_stored(:,ix+1) - u_stored(:,ix) );

  subplot(2,2,1)
  T_i = evalexpr( 'T-273.15', rz, fea );
  plot( rz(1,:), T_i ), hold on, grid on
  text( rz(1,end), T_i(end), [num2str(t_i),' s'] )
  title('Temperature [deg C]')
  axis tight

  subplot(2,2,2)
  sr_i = evalexpr( ['(',sr,')*1e-6'], rz, fea );
  plot( rz(1,:), sr_i ), hold on, grid on
  text( rz(1,end-floor(np/4)), sr_i(end-floor(np/4)), [num2str(t_i),' s'] )
  title('radial stress [MPa]')
  axis tight

  subplot(2,2,3)
  st_i = evalexpr( ['(',st,')*1e-6'], rz, fea );
  plot( rz(1,:), st_i ), hold on, grid on
  text( rz(1,end-floor(np/4)), st_i(end-floor(np/4)), [num2str(t_i),' s'] )
  title('tangential stress [MPa]')
  axis tight

  subplot(2,2,4)
  svm_i = evalexpr( ['(',svm,')*1e-6'], rz, fea );
  plot( rz(1,:), svm_i ), hold on, grid on
  text( rz(1,end-floor(np/4)), svm_i(end-floor(np/4)), [num2str(t_i),' s'] )
  title('von Mieses stress [MPa]')
  axis tight
end
fea.sol.u = u_stored;

figure
hold on

npth = 72;
th   = linspace( 0, 2*pi, npth );

x = opt.rd*cos(th);
z = opt.rd*sin(th);
y = opt.deltad*ones(size(x));
```

```
  plot3(x, y,z,'k-')
  plot3(x,-y,z,'k-')

  x = opt.Rd*cos(th);
  z = opt.Rd*sin(th);
  plot3(x, y,z,'k-')
  plot3(x,-y,z,'k-')

  npr = 25;
  g = ringgrid( npr-1, npth-1, opt.rd, opt.Rd );
  r = sqrt( g.p(1,:).^2 + g.p(2,:).^2 );
  T = evalexpr( 'T-273.15', [r;zeros(size(r))], fea );

  h = patch( 'faces', g.c', 'vertices', [g.p(1,:)' zeros(size(g.p,2),1) g.p(2,:)'], 'facevertexcd
  postplot( fea, 'surfexpr', 'T-273.15', 'colorbar', 'off' )
  fea.grid.p(2,:) = -fea.grid.p(2,:);
  postplot( fea, 'surfexpr', 'T-273.15', 'colorbar', 'off' )
  fea.grid.p(2,:) = -fea.grid.p(2,:);
  view(3)
  axis off
  axis tight
end


out = [];
[Tmin,Tmax] = minmaxsubd( 'T-273.15', fea );
Terr = [ abs((34.179-Tmin)/34.179) abs((88.224-Tmax)/88.224) ];

[stmin,stmax] = minmaxsubd( ['(',st,')*1e-6'], fea );
sterr = [ abs((-21.75-stmin)/-21.75) abs((45.1-stmax)/45.1) ];

[svmmin,svmmax] = minmaxsubd( ['(',svm,')*1e-6'], fea );
svmerr = [ abs((4.478-svmmin)/4.478) abs((46.4-svmmax)/46.4) ];

out.T      = [Tmin Tmax];
out.Terr   = Terr;
out.st     = [stmin stmax];
out.sterr  = sterr;
out.sv     = [svmmin svmmax];
out.svmerr = svmerr;
out.pass   = ~any( [ Terr>0.12 sterr>0.03 svmerr>[0.3 0.01] ] );

if( nargout==0 )
  clear fea out
end
```

11.154 examples/ex_poisson1

EX_POISSON1 1D Poisson equation example.

[FEA, OUT] =

EX_POISSON1(VARARGIN) Poisson equation on a line with a constant source term equal to 1, homogenous boundary conditions, and exact solution (-x^2+x)/2. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {1/10}          Grid cell size
sfun        string {sflag1}        Finite element shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
                                   or use core assembly functions        (<0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'hmax',   1/10;
             'sfun',   'sflag1';
             'refsol', '(-x^2+x)/2';
             'fsrc',   '1';
             'iphys',  1;
             'icub',   2;
             'iplot',  1;
             'tol',    2e-2;
             'fid',    1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Grid generation.
 if( opt.hmax>0 )
   nx = round( 1/opt.hmax );
   fea.grid = linegrid( nx, 0, 1 );
 else   % Scrambled testing grid.
   fea.grid.p = [ 0 1/10 4/10 1/3 1 1-1/3 ];
```

```
    fea.grid.c = [ 1 4 2 6 3 ;
                   2 3 4 5 6 ];
    fea.grid.a = [ 0 2 1 4 3 ;
                   2 4 3 0 5 ];
    fea.grid.b = [ 1 1 1 -1 ;
                   4 2 2  1 ]';
    fea.grid.s = ones(1,5);
  end
  n_bdr = 2;   % Number of boundaries.


% Problem definition.
  fea.sdim  = { 'x' };                    % Coordinate name.
  switch opt.iphys

   case 0   % Directly define fea.eqn/bdr fields.

    fea.dvar  = { 'u' };                  % Dependent variable name.
    fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
    fea.eqn.a.form = { [2;2] };           % First row indicates test function space
(2=x-derivative),
% second row indicates trial function space (2=x-derivative).
    fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.

    fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
    fea.eqn.f.coef = { opt.fsrc };        % Coefficient used in right hand side.

% Define boundary conditions.
    if( strcmp(opt.sfun(end-1:end),'H3') )   % Prescribed derivatives at end points for Hermite ele
      fea.bdr.d = {{ 0 0 ; 1/2 -1/2 }};
    else
      fea.bdr.d     = cell(1,n_bdr);
      [fea.bdr.d{:}] = deal(0);              % Assign zero to all boundaries (homogenous Dirichlet
    end

    fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

% Parse and solve problem.
    fea       = parseprob(fea);           % Check and parse problem struct.
    fea.sol.u = solvestat(fea,'fid',fid,'icub',opt.icub); % Call to stationary solver.

   case 1   % Use physics mode.

    fea = addphys(fea,@poisson);          % Add Poisson equation physics mode.
    fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
    fea.phys.poi.eqn.coef{3,4} = { opt.fsrc };   % Set source term coefficient.
    fea.phys.poi.bdr.coef{1,end} = repmat({0},1,n_bdr);   % Set Dirichlet boundary coefficient to z
    fea = parsephys(fea);                 % Check and parse physics modes.
    if( strcmp(opt.sfun(end-1:end),'H3') )   % Prescribed derivatives at end points for Hermite ele
      fea.bdr.d = {{ 0 0 ; 1/2 -1/2 }};
```

```
    end

% Parse and solve problem.
    fea       = parseprob(fea);           % Check and parse problem struct.
    fea.sol.u = solvestat(fea,'fid',fid,'icub',opt.icub); % Call to stationary solver.

  otherwise   % Use core assembly functions.

    fea.dvar  = { 'u' };                  % Dependent variable name.
    fea.sfun  = { opt.sfun  };            % Shape function.
    fea       = parseprob(fea);           % Check and parse problem struct.

% Assemble stiffness matrix.
    form  = [2;2];
    sfun  = {opt.sfun;opt.sfun};
    coefa = 1;
    sind  = 1;
    i_cub = opt.icub;

    [vRowInds,vColInds,vAvals,n_rows,n_cols] = ...
      assemblea(form,sfun,coefa,i_cub,fea.grid.p,fea.grid.c,fea.grid.a,fea.grid.s,[]);
    A = sparse(vRowInds,vColInds,vAvals,n_rows,n_cols);

% Check and compare with finite difference stencil.
    if (strcmp(opt.sfun,'sflag1'))
      h = 1/nx;
      n = nx+1;
      e = ones(n,1);
      A_ref = 1/h*spdiags([-e 2*e -e], -1:1, n, n);
      A_ref(1) = A_ref(1)/2;
      A_ref(end) = A_ref(end)/2;

      err = norm(A(:)-A_ref(:));
      if err>opt.tol
        out.err  = err;
        out.pass = -1;
        return
      end
    end

    form  = 1;
    sfun  = sfun{1};
    coeff = 1;

    f = assemblef(form,sfun,coeff,i_cub,fea.grid.p,fea.grid.c,fea.grid.a,fea.grid.s,[]);


% Check and compare with finite difference stencil.
    if (strcmp(opt.sfun,'sflag1'))
      f_ref          = coeff*h*ones(n,1);
      f_ref([1 end]) = coeff*1/2*h;
```

```
      err = norm(f-f_ref);
      if err>1e-6
        out.err  = err;
        out.pass = -2;
        return
      end
    end

% Set homogenous Dirichlet boundary conditions on first and last dof/node.
    bind = [1 nx+1];
    A = A';   %'
    A(:,bind) = 0;          % Zero out Dirichlet BC rows.
    for i=1:length(bind)    % Loop to set diagonal entry to 1.
      i_a = bind(i);
      A(i_a,i_a) = 1;
    end
    A = A';   %'
    f(bind) = 0;            % Set corresponding source term entries to Dirichlet BC values.

% Solve problem.
    fea.sol.u = A\f;

 end


% Postprocessing.
 if ( opt.iplot>0 )
    x = linspace( 0, 1, 41 );
    u = evalexpr( 'u', x, fea )';
    figure
    subplot(3,1,1)
    plot( x, u )
    axis( [0 1 0 0.2])
    grid on
    title('Solution u')
    subplot(3,1,2)
    ux = (-x.^2+x)/2;
    plot( x, ux )
    axis( [0 1 0 0.2])
    grid on
    title('Exact solution')
    subplot(3,1,3)
    plot( x, abs(ux-u) )
    title('Error')
 end


% Error checking.
 xi = [1/2; 1/2];
 s_err = ['abs(',opt.refsol,'-u)'];
 err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
 ref = evalexpr0('u',xi,1,1:size(fea.grid.c,2),[],fea);
```

```
  err = sqrt(sum(err.^2)/sum(ref.^2));

  if( ~isempty(fid) )
    fprintf(fid,'\nL2 Error: %e\n',err)
    fprintf(fid,'\n\n')
  end

  out.err  = err;
  out.tol  = opt.tol;
  out.pass = out.err<out.tol;
  if ( nargout==0 )
    clear fea out
  end
```

11.155 examples/ex_poisson2

EX_POISSON2 2D Poisson equation example on a circle.

[FEA, OUT] =

EX_POISSON2(VARARGIN) Poisson equation on a circle with source term 1 and exact solution u=(1-r^2)/4. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
radi        scalar {1}             Radius of circle
hmax        scalar {0.1}           Max grid cell size
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
igrid       scalar -1/{0}          Cell type (0>=triangles, 0<quadrilaterals)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
  cOptDef = { ...
    'radi',     1; ...
```

```
    'hmax',     0.1; ...
    'refsol',   '(1-(x^2+y^2))/4'; ...
    'fsrc',     '1'; ...
    'sfun',     'sflag1'; ...
    'iphys',    1; ...
    'icub',     2; ...
    'iplot',    1; ...
    'igrid',    1; ...
    'tol',      1e-1;
    'fid',      1 };
  [got,opt] = parseopt(cOptDef,varargin{:});
  fid       = opt.fid;


% Geometry definition.
  gobj = gobj_circle( [0 0], opt.radi );
  fea.geom.objects = { gobj };


% Grid generation.
  if( opt.igrid<0 )
    fea.grid = circgrid(4,3);
  else
    fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
  end
  n_bdr = max(fea.grid.b(3,:));        % Number of boundaries.


% Problem definition.
  fea.sdim = { 'x' 'y' };                % Coordinate names.
  if ( opt.iphys==1 )

    fea = addphys(fea,@poisson);          % Add Poisson equation physics mode.
    fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
    fea.phys.poi.eqn.coef{3,4} = { opt.fsrc };   % Set source term coefficient.
    fea.phys.poi.bdr.coef{1,end} = repmat({opt.refsol},1,n_bdr);   % Set Dirichlet boundary coeffic
%   fea.phys.poi.eqn
    fea = parsephys(fea);                 % Check and parse physics modes.
    if( any(strcmp(opt.sfun,{'sf_tri_H3','sf_quad_H3'})) )   % Prescribed derivatives at end points
      fea.bdr.d = {{ opt.refsol opt.refsol opt.refsol opt.refsol ;
                     '-x/2' '-x/2' '-x/2' '-x/2' ;
                     '-y/2' '-y/2' '-y/2' '-y/2' }};
    end

  else

    fea.dvar  = { 'u' };                  % Dependent variable name.
    fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
    fea.eqn.a.form = { [2 3;2 3] };       % First row indicates test function space
(2=x-derivative + 3=y-derivative),
```

```
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
   fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.

   fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
   fea.eqn.f.coef = { opt.fsrc };        % Coefficient used in right hand side.

% Define boundary conditions.
   if( any(strcmp(opt.sfun,{'sf_tri_H3','sf_quad_H3'})) )   % Prescribed derivatives at end points
     fea.bdr.d = {{ opt.refsol opt.refsol opt.refsol opt.refsol ;
                    '-x/2' '-x/2' '-x/2' '-x/2' ;
                    '-y/2' '-y/2' '-y/2' '-y/2' }};
   else
     fea.bdr.d     = cell(1,n_bdr);
    [fea.bdr.d{:}] = deal(opt.refsol);   % Assign reference solution to all boundaries (Dirichlet)
   end

   fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid,'icub',opt.icub);   % Call to stationary solver.


% Postprocessing.
 s_err = ['abs(',opt.refsol,'-u)'];
 if ( opt.iplot>0 )
   figure
   subplot(3,1,1)
   postplot(fea,'surfexpr','u','axequal','on')
   title('Solution u')
   subplot(3,1,2)
   postplot(fea,'surfexpr',opt.refsol,'axequal','on')
   title('Exact solution')
   subplot(3,1,3)
   postplot(fea,'surfexpr',s_err,'axequal','on','evalstyle','exact')
   title('Error')
 end


% Error checking.
 if ( size(fea.grid.c,1)==4 )
   xi = [0;0];
 else
   xi = [1/3;1/3;1/3];
 end
 err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
 ref = evalexpr0('u',xi,1,1:size(fea.grid.c,2),[],fea);
 err = sqrt(sum(err.^2)/sum(ref.^2));
```

```
  if( ~isempty(fid) )
    fprintf(fid,'\nL2 Error: %f\n',err)
    fprintf(fid,'\n\n')
  end


  out.err  = err;
  out.tol  = opt.tol;
  out.pass = out.err<out.tol;
  if ( nargout==0 )
    clear fea out
  end
```

11.156 examples/ex_poisson3

EX_POISSON3 2D Poisson equation example on a unit square.

[FEA, OUT] =
EX_POISSON3(VARARGIN) Poisson equation on a
[0..1]^2 unit square.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.1}           Max grid cell size
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
  cOptDef = { ...
    'igrid',    0; ...
    'hmax',     0.1; ...
    'sfun',     'sflag1'; ...
```

```
    'iphys',    1; ...
    'icub',     2; ...
    'iplot',    1; ...
    'tol',      0.01;
    'fid',      1 };
  [got,opt] = parseopt(cOptDef,varargin{:});
  fid       = opt.fid;


% Geometry definition.
  fea.geom.objects = { gobj_rectangle() };


% Grid generation.
  switch opt.igrid
    case -2
      n  = round(1/opt.hmax);
      fea.grid = rectgrid(n,n,[0 1;0 1]);
      ix = setdiff( 1:(n+1)^2, [1:(n+1) (n+1):(n+1):(n+1)^2 (n+1)^2:-1:(n+1)^2-(n+1)+1 (n+1)^2-(n+1
      h  = 0.3*1/n;
      fea.grid.p(1,ix) = fea.grid.p(1,ix) + h*(2*rand(1,numel(ix)) - 1);
      fea.grid.p(2,ix) = fea.grid.p(2,ix) + h*(2*rand(1,numel(ix)) - 1);
    case -1
      fea.grid = rectgrid(round(1/opt.hmax),round(1/opt.hmax),[0 1;0 1]);
      fea.grid = quad2tri(fea.grid);
    case 0
      fea.grid = rectgrid(round(1/opt.hmax),round(1/opt.hmax),[0 1;0 1]);
    case 1
      fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
  end
  n_bdr = max(fea.grid.b(3,:));           % Number of boundaries.


% Problem definition.
  fea.sdim  = { 'x' 'y' };                % Coordinate names.
  if ( opt.iphys==1 )

    fea = addphys(fea,@poisson);          % Add Poisson equation physics mode.
    fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
    fea.phys.poi.eqn.coef{3,4} = { 1 };   % Set source term coefficient.
    fea.phys.poi.bdr.coef{1,end} = repmat({0},1,n_bdr);   % Set Dirichlet boundary coefficient to z
    fea = parsephys(fea);                 % Check and parse physics modes.

    if( any(strcmp(opt.sfun,{'sf_tri_H3','sf_quad_H3'})) )   % Prescribed derivatives at end points
      fea.bdr.d = {{ 0 0 0 0 ;
                     0 '-ex_poisson3_derivative(y)' 0 'ex_poisson3_derivative(y)' ;
                     'ex_poisson3_derivative(x)' 0 '-ex_poisson3_derivative(x)' 0 }};
    end

  else

    fea.dvar  = { 'u' };                  % Dependent variable name.
```

```
    fea.sfun  = { opt.sfun };             % Shape function.

% Define equation system.
    fea.eqn.a.form = { [2 3;2 3] };       % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
    fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.

    fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
    fea.eqn.f.coef = { 1 };               % Coefficient used in right hand side.

% Define boundary conditions.
    if( any(strcmp(opt.sfun,{'sf_tri_H3','sf_quad_H3'})) )   % Prescribed derivatives at end points
      fea.bdr.d = {{ 0 0 0 0 ;
                     0 '-ex_poisson3_derivative(y)' 0 'ex_poisson3_derivative(y)' ;
                     'ex_poisson3_derivative(x)' 0 '-ex_poisson3_derivative(x)' 0 }};
    else
      fea.bdr.d     = cell(1,n_bdr);
     [fea.bdr.d{:}] = deal(0);          % Assign zero to all boundaries (Dirichlet).
    end
    fea.bdr.n     = cell(1,n_bdr);    % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid,'icub',opt.icub);   % Call to stationary solver.


% Postprocessing.
 if( opt.iplot>0 )
    figure
    g = rectgrid( 20 );
    u = evalexpr( 'u', g.p, fea );
    fv.faces    = g.c';
    fv.vertices = [g.p' u];
    fv.facevertexcdata = u;
    fv.facecolor = 'interp';
    patch( fv )
    grid on, axis normal, view(3)
    xlabel( 'x' )
    ylabel( 'y' )
    title('Solution u')
 end


% Error checking.
 x = linspace( 0, 1, 11 );
 [x,y] = meshgrid(x,x);
 u = evalexpr( 'u', [x(:) y(:)]', fea );
 u_ref = l_poisol2( x(:), y(:), 6 );
```

```
  u_diff = u - u_ref;
  err = norm(u_diff);
  if( ~isempty(fid) )
    fprintf(fid,'\nL2  Error: %f\n',err)
    fprintf(fid,'\nL00 Error: %f\n',max(abs(u_diff)))
    fprintf(fid,'\n\n')
  end


  out.err  = err;
  out.tol  = opt.tol;
  out.pass = out.err<opt.tol;
  if ( nargout==0 )
    clear fea out
  end


%   ------------------------------------------------------------------------
```

11.157 examples/ex_poisson3_derivative

EX_POISSON3_DERIVATIVE Derivative interpolation function.

[UX_X] =
EX_POISSON3_DERIVATIVE(X) Derivative interpolation function

Code listing

```
  ux   = [0 0.107535709861634 0.170943549921419 0.217687143600172 0.253587789782147 0.2813834882560
  ux_x = interp1( linspace(0,1,numel(ux)), ux, x, 'pchip' );
  ux_x( x<=0 | x>=1 ) = 0;
```

11.158 examples/ex_poisson4

EX_POISSON4 2D Poisson equation example on a rectangle.

[FEA, OUT] =
EX_POISSON4(VARARGIN) Poisson equation on a rectangle with source term (1+2000^2)

∗ pi^2 ∗ sin(pi∗x) ∗ sin(2000∗pi∗y) and exact solution sin(pi∗x) ∗ sin(2000∗pi∗y). Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
lx          scalar {1e-3}          Length of domain in x-direction
ly          scalar {1e-3}          Length of domain in y-direction
hmax        scalar {ly/20}         Max grid cell size
refsol      string {sin(pi*x)*sin(2000*pi*y)}   Reference solution
fsrc        string {(1+2000^2)*pi^2*sin(pi*x)*sin(2000*pi*y)}   Source term
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'igrid',    0; ...
   'lx',       0.001; ...
   'ly',       0.001; ...
   'hmax',     0.001/20; ...
   'refsol',   'sin(pi*x)*sin(2000*pi*y)'; ...
   'fsrc',     '(1+2000^2)*pi^2*sin(pi*x)*sin(2000*pi*y)'; ...
   'sfun',     'sflag1'; ...
   'iphys',    1; ...
   'iplot',    1; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 gobj = gobj_rectangle( 0, opt.lx, 0, opt.lx );
 fea.geom.objects = { gobj };


% Grid generation.
 switch opt.igrid
   case -1
```

```
      fea.grid = rectgrid(round(opt.lx/opt.hmax),round(opt.lx/opt.hmax),[0 opt.lx;0 opt.lx]);
      fea.grid = quad2tri(fea.grid);
    case 0
      fea.grid = rectgrid(round(opt.lx/opt.hmax),round(opt.lx/opt.hmax),[0 opt.lx;0 opt.lx]);
    case 1
      fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 end
 fea.grid.p(2,:) = fea.grid.p(2,:)*opt.ly/opt.lx;   % Scale y-coordinates.
 n_bdr = max(fea.grid.b(3,:));           % Number of boundaries.


% Problem definition.
 fea.sdim  = { 'x' 'y' };                % Coordinate names.
 if ( opt.iphys==1 )

   fea = addphys(fea,@poisson);          % Add Poisson equation physics mode.
   fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
   fea.phys.poi.eqn.coef{3,4} = { opt.fsrc };   % Set source term coefficient.
   fea.phys.poi.bdr.coef{1,end} = repmat({opt.refsol},1,n_bdr);   % Set Dirichlet boundary coeffic
   fea = parsephys(fea);                 % Check and parse physics modes.

 else

   fea.dvar  = { 'u' };                  % Dependent variable name.
   fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
   fea.eqn.a.form = { [2 3;2 3] };       % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
   fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.

   fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
   fea.eqn.f.coef = { opt.fsrc };        % Coefficient used in right hand side.

% Define boundary conditions.
   fea.bdr.d     = cell(1,n_bdr);
  [fea.bdr.d{:}] = deal(opt.refsol);     % Assign reference solution to all boundaries (Dirichlet)

   fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid);   % Call to stationary solver.


% Postprocessing.
 s_err = ['abs(',opt.refsol,'-u)'];
 if ( opt.iplot>0 )
```

```
    figure
    subplot(3,1,1)
    postplot(fea,'surfexpr','u','axequal','off')
    title('Solution u')
    subplot(3,1,2)
    postplot(fea,'surfexpr',opt.refsol,'axequal','off')
    title('Exact solution')
    subplot(3,1,3)
    postplot(fea,'surfexpr',s_err,'axequal','off','evalstyle','exact')
    title('Error')
  end


% Error checking.
 if ( size(fea.grid.c,1)==4 )
   xi = [0;0];
 else
   xi = [1/3;1/3;1/3];
 end
 err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
 ref = evalexpr0('u',xi,1,1:size(fea.grid.c,2),[],fea);
 err = sqrt(sum(err.^2)/sum(ref.^2));

 if( ~isempty(fid) )
   fprintf(fid,'\nL2 Error: %f\n',err)
   fprintf(fid,'\n\n')
 end


 out.err  = err;
 out.pass = out.err<0.1;
 if ( nargout==0 )
   clear fea out
 end
```

11.159 examples/ex_poisson5

EX_POISSON5 3D Poisson equation example on a unit sphere.

[FEA, OUT] =

EX_POISSON5(VARARGIN) Poisson equation on a unit sphere with source term f=1 and homogenous (zero) Dirichlet boundary conditions on the sphere surface. The exact solution to this problem is u_ref=(1-r^2)/6 where r is the radius from the origin. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
igrid       scalar 1/{0}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.35}          Max grid cell size
sfun        string {sflag1}        Shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'igrid',    0; ...
   'hmax',     0.35; ...
   'refsol',   '(1-(x^2+y^2+z^2))/6'; ...
   'sfun',     'sflag1'; ...
   'iphys',    1; ...
   'icub',     2; ...
   'iplot',    1; ...
   'tol',      0.2;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 gobj = gobj_sphere();
 fea.geom.objects = { gobj };


% Grid generation.
 switch opt.igrid
   case -2
     fea.grid = blockgrid( 1, 1, 1, [-1 1;-1 1;-1 1] );
   case -1
     fea.grid = spheregrid(round(1/opt.hmax*4/7),round(1/opt.hmax*3/7));
     fea.grid = hex2tet(fea.grid);
   case 0
     fea.grid = spheregrid(max(2,5*round(1/opt.hmax*4/7)),max(1,5*round(1/opt.hmax*3/7)));
   case 1
     fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 end
 n_bdr = max(fea.grid.b(3,:));           % Number of boundaries.
```

```
% Problem definition.
 fea.sdim  = { 'x' 'y' 'z' };            % Coordinate names.
 if ( opt.iphys==1 )

   fea = addphys(fea,@poisson);          % Add Poisson equation physics mode.
   fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
   fea.phys.poi.eqn.coef{3,4} = { 1 };   % Set source term coefficient.
   fea.phys.poi.bdr.coef{1,end} = repmat({opt.refsol},1,n_bdr);   % Assign reference solution to a
   fea = parsephys(fea);                 % Check and parse physics modes.

 else

   fea.dvar  = { 'u' };                  % Dependent variable name.
   fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
   fea.eqn.a.form = { [2 3 4;2 3 4] };   % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
   fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.

   fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
   fea.eqn.f.coef = { 1 };               % Coefficient used in right hand side.

% Define boundary conditions.
   fea.bdr.d     = cell(1,n_bdr);
  [fea.bdr.d{:}] = deal(opt.refsol);     % Assign reference solution to all boundaries (Dirichlet)

   fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid,'icub',opt.icub);   % Call to stationary solver.


% Postprocessing.
 if ( opt.iplot>0 )
   figure
   postplot(fea,'surfexpr','u','selexpr','(y>0)','axequal','on')
 end


% Error checking.
 s_err = ['abs(',opt.refsol,'-u)'];
 if ( size(fea.grid.c,1)==8 )
   xi = [0;0;0];
 else
   xi = [1/4;1/4;1/4;1/4];
```

```
  end
  err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
  ref = evalexpr0('u',xi,1,1:size(fea.grid.c,2),[],fea);
  err = sqrt(sum(err.^2)/sum(ref.^2));

  if( ~isempty(fid) )
    fprintf(fid,'\nL2 Error: %f\n',err)
    fprintf(fid,'\n\n')
  end


  out.err  = err;
  out.tol  = opt.tol;
  out.pass = out.err<out.tol;
  if ( nargout==0 )
    clear fea out
  end
```

11.160 examples/ex_poisson6

EX_POISSON6 3D Poisson equation example on a unit cube.

[FEA, OUT] =

EX_POISSON6(VARARGIN) Poisson equation on a [0..1]^3 unit cube.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
igrid       scalar 1/{0}           Cell type (0=hexahedra, 1=tetrahedra)
hmax        scalar {1/9}           Max grid cell size
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'igrid',    0; ...
   'hmax',     1/9; ...
   'sfun',     'sflag1'; ...
   'iphys',    1; ...
   'icub',     2; ...
   'iplot',    1; ...
   'tol',      0.05; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 gobj = gobj_block();
 fea.geom.objects = { gobj };


% Grid generation.
 switch opt.igrid
   case -1
     fea.grid = blockgrid(round(1/opt.hmax));
     fea.grid = hex2tet(fea.grid);
   case 0
     fea.grid = blockgrid(round(1/opt.hmax));
   case 1
     fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 end
 n_bdr = max(fea.grid.b(3,:));           % Number of boundaries.


% Problem definition.
 fea.sdim  = { 'x' 'y' 'z' };            % Coordinate names.
 if ( opt.iphys==1 )

   fea = addphys(fea,@poisson);          % Add Poisson equation physics mode.
   fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
   fea.phys.poi.eqn.coef{3,4} = { 1 };   % Set source term coefficient.
   fea.phys.poi.bdr.coef{1,end} = repmat({0},1,n_bdr);   % Set Dirichlet boundary coefficient to z
   fea = parsephys(fea);                 % Check and parse physics modes.

 else

   fea.dvar  = { 'u' };                  % Dependent variable name.
   fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
   fea.eqn.a.form = { [2 3 4;2 3 4] };   % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
   fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.
```

```
   fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
   fea.eqn.f.coef = { 1 };               % Coefficient used in right hand side.

% Define boundary conditions.
   fea.bdr.d     = cell(1,n_bdr);
  [fea.bdr.d{:}] = deal(0);              % Assign zero to all boundaries (Dirichlet).

   fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Parse and solve problem.
 fea       = parseprob(fea);             % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',fid,'icub',opt.icub);   % Call to stationary solver.


% Postprocessing.
 if ( opt.iplot>0 )
   p      = l_xgrid(fea.grid,opt.sfun);
   u_ref  = refsol_poi3dcube(p(:,1),p(:,2),p(:,3),3);
   figure
   subplot(1,2,1)
   postplot(fea,'surfexpr','u','selexpr','(y>0.5)','axequal','on')
   title('Solution u')
   subplot(1,2,2)
   postplot(fea,u_ref(1:size(fea.grid.p,2)),'surfexpr','u','selexpr','(y>0.5)','axequal','on')
   title('Reference Solution')
 end


% Error checking.
 x = linspace( 0+sqrt(eps), 1-sqrt(eps), 11 );
 [x,y,z] = ndgrid(x,x,x);
 u = evalexpr( 'u', [x(:) y(:) z(:)]', fea );
 u_ref = refsol_poi3dcube( x(:), y(:), z(:), 6 );
 err = sqrt(sum((u-u_ref).^2)/sum(u_ref.^2));
 if( ~isempty(fid) )
   fprintf(fid,'\nL2 Error: %f\n',err)
   fprintf(fid,'\n\n')
 end


 out.err  = err;
 out.pass = out.err<opt.tol;
 if ( nargout==0 )
   clear fea out
 end


%-----------------------------------------------
```

11.161 examples/ex_poisson7

EX_POISSON7 Poisson equation on a unit circle with a point source.

[FEA, OUT] =

EX_POISSON7(VARARGIN) Poisson equation on a unit circle with a point source (represented by a point constraint) and exact solution u = -1/(2∗pi)∗log(r). Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {0.1}           Max grid cell size (<0 quadrilateral grid)
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'hmax',     0.02;
   'refsol',   '-1/(2*pi)*log(sqrt(x^2+y^2))';
   'sfun',     'sflag1';
   'iphys',    1;
   'iplot',    1;
   'tol',      0.2;
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
 fea.geom.objects = { gobj_circle() };
```

```
% Grid generation.
 if( opt.hmax<0 )
   fea.grid = circgrid(10);
 else
   hmax      = opt.hmax;
   fh        = @(p,varargin) 3*hmax + (p(:,1).^2+p(:,2).^2);
   fea.grid  = gridgen( fea, 'hmax', hmax, 'fid', fid, 'fixpnt', [0 0], 'hdfcn', fh ) ;
 end


% Problem definition.
 fea.sdim  = { 'x' 'y' };                % Coordinate names.
 if( opt.iphys==1 )

   fea = addphys( fea, @poisson );       % Add Poisson equation physics mode.
   fea.phys.poi.sfun = { opt.sfun };     % Set shape function.
   fea.phys.poi.eqn.coef{3,4} = { 0 };   % Set source term coefficient.
   fea = parsephys(fea);                 % Check and parse physics modes.

 else

   fea.dvar  = { 'u' };                  % Dependent variable name.
   fea.sfun  = { opt.sfun  };            % Shape function.

% Define equation system.
   fea.eqn.a.form = { [2 3;2 3] };       % First row indicates test function space
(2=x-derivative + 3=y-derivative),
% second row indicates trial function space (2=x-derivative + 3=y-derivative).
   fea.eqn.a.coef = { 1 };               % Coefficient used in assembling stiffness matrix.

   fea.eqn.f.form = { 1 };               % Test function space to evaluate in right hand side (1=f
   fea.eqn.f.coef = { 0 };               % Coefficient used in right hand side.

% Define boundary conditions.
   n_bdr         = max(fea.grid.b(3,:));
   fea.bdr.d     = cell(1,n_bdr);
  [fea.bdr.d{:}] = deal(0);              % Assign zero to all boundaries (Dirichlet).

   fea.bdr.n     = cell(1,n_bdr);        % No Neumann boundaries ('fea.bdr.n' empty).

 end


% Set point constraint.
 [~,i_mid] = min( fea.grid.p(1,:).^2 + fea.grid.p(2,:).^2 );
 fea.pnt.index = i_mid;
 fea.pnt.type  = 'source';
 fea.pnt.dvar  = 1;
 fea.pnt.expr  = 1;


% Parse and solve problem.
```

```
  fea       = parseprob( fea );               % Check and parse problem struct.
  fea.sol.u = solvestat( fea, 'fid', fid );   % Call to stationary solver.


% Postprocessing.
 s_err = ['abs(',opt.refsol,'-u)'];
 if( opt.iplot>0 )
   figure
   subplot(3,1,1)
   postplot( fea, 'surfexpr', 'u', 'surfhexpr', 'u', 'axequal', 'on' )
   title('Solution u')

   subplot(3,1,2)
   postplot( fea, 'surfexpr', opt.refsol, 'surfhexpr', opt.refsol, 'axequal', 'on' )
   title('Exact solution')

   subplot(3,1,3)
   postplot( fea, 'surfexpr', s_err, 'surfhexpr', s_err, 'axequal', 'on' )
   title('Error')
 end


% Error checking.
 if ( size(fea.grid.c,1)==4 )
   xi = [0;0];
 else
   xi = [1/3;1/3;1/3];
 end
 err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
 ref = evalexpr0('u',xi,1,1:size(fea.grid.c,2),[],fea);
 err = sqrt(sum(err.^2)/sum(ref.^2));

 if( ~isempty(fid) )
   fprintf(fid,'\nL2 Error: %f\n',err)
   fprintf(fid,'\n\n')
 end


 out.err  = err;
 out.pass = out.err<opt.tol;
 if( nargout==0 )
   clear fea out
 end
```

11.162 examples/ex_periodic1

EX_PERIODIC1 1D Example of pulse in a periodic line.

[FEA, OUT] =

EX_PERIODIC1(VARARGIN) Moving 1D pulse in a periodic domain. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
init        string <tt>(x>=0.4)*(x<=0.6)</tt>   Initial shape of pulse
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'init',     '(x>=0.4)*(x<=0.6)';
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;

 fea.sdim = {'x'};
 fea.grid = linegrid( 100 );

 fea = addphys( fea, @convectiondiffusion );
 fea.phys.cd.eqn.coef{3,end} = { 1 };
 fea.phys.cd.eqn.coef{2,end} = { 0.002 };
 fea = parsephys( fea );

 fea.bdr.d{1} = { [] [] };
 fea.bdr.n{1}{2} = 'solve_hook_periodic_bc';


% Save pediodic boundary function to file.
 s_file = fileread( [mfilename('fullpath'),''] );
 ix = strfind( s_file, 'function' );
 s_fcn = s_file(ix(end):end);
 ix1 = find( s_fcn == '=', 1 );
 ix2 = find( s_fcn == '(', 1 );
 s_fname = strtrim(s_fcn(ix1+1:ix2-1));
 fid_tmp = fopen( [s_fname,''], 'w' );
 fprintf( fid_tmp, '%s', s_fcn );
 fclose( fid_tmp );
```

```
% Parse and solve problem.
 fea = parseprob( fea );
 fea.sol.u = solvetime( fea, 'init', opt.init, 'tstep', 0.005, 'tmax', 1, 'fid', opt.fid );


% Postprocessing.
 if( opt.iplot>0 )
   postplot( fea, 'surfexpr', 'c', 'solnum', 1, 'color', 'b' )
   postplot( fea, 'surfexpr', 'c', 'solnum', floor(size(fea.sol.u,2)/4), 'color', 'c' )
   postplot( fea, 'surfexpr', 'c', 'solnum', floor(size(fea.sol.u,2)/2), 'color', 'm' )
   postplot( fea, 'surfexpr', 'c', 'solnum', floor(size(fea.sol.u,2)*3/4), 'color', 'g' )
   postplot( fea, 'surfexpr', 'c', 'solnum', size(fea.sol.u,2) )
   axis( [0 1 0 1] )
   grid on
 end


% Error checking.
 if( strcmp(opt.init,'(x>=0.4)*(x<=0.6)') )
   i_ref  = 0.2;
   i_calc = intsubd( 'c*(x>=0.4)*(x<=0.6)', fea );
   err = abs(i_calc-i_ref)/i_ref;

   if( ~isempty(fid) )
     fprintf(fid,'\nError: %f\n',err)
     fprintf(fid,'\n\n')
   end
 end


 out.err  = err;
 out.pass = err<0.3;
 if ( nargout==0 )
   clear fea out
 end


%
```

11.163 examples/ex_periodic2

EX_PERIODIC2 2D Periodic Poisson equation example.

[FEA, OUT] =

EX_PERIODIC2(VARARGIN) 2D Periodic Poisson equation example. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
isol        scalar 0/{1}           Stationary/Time dependent solution
iplot       scalar 0/{1}           Plot solution and error (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output struct
```

Code listing

```
 cOptDef = { 'isol',     0;
             'iplot',    1;
             'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Unit square grid.
 fea.sdim = {'x' 'y'};
 fea.grid = rectgrid( 40 );


% Define Poisson equation with custom physics mode.
 fea = addphys( fea, @customeqn, { 'u' } );
 fea.phys.ce.eqn.seqn = '- ux_x - uy_y = f + u*eps';


% Boundary conditions.
 DirBC = true;
 NeuBC = ~DirBC;
 NeuBC_coef = 'solve_hook_periodic_bc';
 fea.phys.ce.bdr.coef = { 'bcnd_ce', '', '', {}, ...
                          { DirBC, NeuBC,      DirBC, NeuBC }, [], ...
                          { 0    , NeuBC_coef, 0,     0     } };

% Source term.
 fea.expr = { 'f' 'x*sin(5*pi*y) + exp(-((x-0.5)^2 + (y-0.5)^2)/0.02)' };


% Save pediodic boundary function to file.
 s_file = fileread( [mfilename('fullpath'),''] );
 ix = strfind( s_file, 'function' );
 s_fcn = s_file(ix(end):end);
 ix1 = find( s_fcn == '=', 1 );
```

```
 ix2 = find( s_fcn == '(', 1 );
 s_fname = strtrim(s_fcn(ix1+1:ix2-1));
 fid_tmp = fopen( [s_fname,''], 'w' );
 fprintf( fid_tmp, '%s', s_fcn );
 fclose( fid_tmp );


% Parse and solve problem.
 fea = parsephys( fea );
 fea = parseprob( fea );
 if( opt.isol==0 )
   fea.sol.u = solvestat( fea, 'fid', opt.fid );
 else
   fea.sol.u = solvetime( fea, 'tstop', -eps, 'fid', opt.fid );
 end


% Postprocessing.
 y  = linspace( 0, 1, 100 );
 ul = evalexpr( 'u', [zeros(1,100);y], fea );
 ur = evalexpr( 'u', [ones(1,100);y],  fea );
 if( opt.iplot>0 )
   subplot(1,2,1)
   postplot(fea, 'surfexpr', 'u', 'surfhexpr', 'u' )

% Plot solution on left and right sides.
   hold on
   fea.grid.p(1,:) = fea.grid.p(1,:) - 1;
   postplot(fea, 'surfexpr', 'u', 'surfhexpr', 'u' )
   fea.grid.p(1,:) = fea.grid.p(1,:) + 2;
   postplot(fea, 'surfexpr', 'u', 'surfhexpr', 'u' )

   subplot(1,2,2)
   plot( y, ul, 'k-' )
   hold on
   plot( y, ur, 'r.' )
   grid on
   xlabel( 'y' )
   ylabel( 'u' )
   legend( 'u(x=0)', 'u(x=1)', 'Location', 'South' )
 end


% Error checking.
 err = norm( ur - ul );

 out.err  = err;
 out.pass = err<1e-6;
 if ( nargout==0 )
   clear fea out
 end
```

```
%
```

11.164 examples/ex_robinbc1

EX_ROBINBC1 1D Robin boundary condition example.

[FEA, OUT] =

EX_ROBINBC1(VARARGIN) Convection, diffusion, and reaction equation uxx+u∗ux-u=exp(2x) on a line with a Robin boundary condition u(0)+ux(0)=2 and u(1)=exp(1), and exact solution exp(x). Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
hmax        scalar {1/10}          Grid cell size
sfun        string {sflag1}        Finite element shape function
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { 'hmax',   1/10;
             'sfun',   'sflag1';
             'iplot',  1;
             'icub',   2;
             'refsol', 'exp(x)';
             'tol',    3e-3;
             'fid',    1 };
 [got,opt] = parseopt(cOptDef,varargin{:});

 nx = round( 1/opt.hmax );

 fea.sdim  = {'x'};
 fea.grid  = linegrid(nx,0,1);
 fea.dvar  = {'u'};
 fea.sfun  = {opt.sfun};
 fea.eqn   = parseeqn( 'ux_x + u*ux_t - u_t = exp(2*x)', ...
```

```
                        fea.dvar, fea.sdim );
 fea.bdr.d = {[],opt.refsol};
 fea.bdr.n = {'2-u',[]};


% Parse and solve problem.
 fea       = parseprob(fea);           % Check and parse problem struct.
 fea.sol.u = solvestat(fea,'fid',opt.fid,'icub',opt.icub); % Call to stationary solver.


% Postprocessing.
 if( opt.iplot>0 )
   h1 = postplot( fea, 'surfexpr', 'u', 'color', 'b' );
   h2 = postplot( fea, 'surfexpr', opt.refsol, ...
                  'linestyle', '--', 'color', 'r' );
   grid on
   legend( [h1(1),h2(1)], 'computed solution', ...
           'analytic solution', 'location', 'northwest' )
   xlabel('x')
   ylabel('u')
   axis normal
 end


% Error checking.
 xi = [1/2; 1/2];
 s_err = ['abs(',opt.refsol,'-u)'];
 err = evalexpr0(s_err,xi,1,1:size(fea.grid.c,2),[],fea);
 ref = evalexpr0('u',xi,1,1:size(fea.grid.c,2),[],fea);
 err = sqrt(sum(err.^2)/sum(ref.^2));

 if( ~isempty(opt.fid) )
   fprintf(opt.fid,'\nL2 Error: %e\n',err)
   fprintf(opt.fid,'\n\n')
 end

 out.err  = err;
 out.tol  = opt.tol;
 out.pass = out.err<out.tol;
 if( nargout==0 )
   clear fea out
 end
```

11.165 examples/ex_waveequation1

EX_WAVEEQUATION1 2D Wave equation example on a circle.

[FEA, OUT] =

EX_WAVEEQUATION1(VARARGIN) Wave equation on a circle with zero source term by variable splitting. Accepts the following property/value pairs.

```
Input       Value/{Default}        Description
-------------------------------------------------------------------------------
radi        scalar {1}             Radius of circle
c2          scalar {1}             Wave speed (squared)
init        scalar {1-(x^2+y^2)}   Initial shape
tmax        scalar {1}             Stopping time
tstep       scalar {0.1}           Time step size
igrid       scalar 0/{1}           Cell type (0=quadrilaterals, 1=triangles)
hmax        scalar {0.05}          Grid cell size
iexpl       scalar {0}             Use explicit (or implicit) right hand side
ischeme     scalar {3}             Time stepping scheme
sfun        string {sflag1}        Shape function
iphys       scalar 0/{1}           Use physics mode to define problem    (=1)
                                   or directly define fea.eqn/bdr fields (=0)
iplot       scalar 0/{1}           Plot solution (=1)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
fea         struct                 Problem definition struct
out         struct                 Output stuct
```

Code listing

```
 cOptDef = { ...
   'radi',     1; ...
   'c2',       1; ...
   'init',     '1-(x^2+y^2)'; ...
   'tmax',     1; ...
   'igrid',    1; ...
   'hmax',     0.1; ...
   'tstep',    0.05; ...
   'iexpl'     0; ...
   'ischeme'   3; ...
   'sfun',     'sflag1'; ...
   'iphys',    1; ...
   'iplot',    1; ...
   'tol',      0.125; ...
   'fid',      1 };
 [got,opt] = parseopt(cOptDef,varargin{:});
 fid       = opt.fid;


% Geometry definition.
```

```
 gobj = gobj_circle( [0 0], opt.radi );
 fea.geom.objects = { gobj };


% Grid generation.
 if( opt.igrid==1 )
   fea.grid = gridgen(fea,'hmax',opt.hmax,'fid',fid);
 else
   fea.grid = circgrid( 16, 12, opt.radi );
   if( opt.igrid<0 )
     fea.grid = quad2tri( fea.grid );
   end
 end
 n_bdr = max(fea.grid.b(3,:));   % Number of boundaries.


% Problem definition.
 fea.sdim  = { 'x' 'y' };   % Coordinate names.
 if ( opt.iphys==1 )

   fea = addphys( fea, @poisson, {'u'} );  % Add Poisson equation physics mode for 'u'.
   fea = addphys( fea, @poisson, {'v'} );  % Add Poisson equation physics mode for 'v'.
   fea.phys.poi.sfun  = { opt.sfun };      % Set shape function for 'u'.
   fea.phys.poi2.sfun = { opt.sfun };      % Set shape function for 'v'.

% Change equations.
   if( opt.iexpl==1 )
     fea.phys.poi.eqn.seqn  = ['u'' = v'];
   else
     fea.phys.poi.eqn.seqn  = ['u'' - v_t = 0'];
   end
   fea.phys.poi2.eqn.seqn = ['v'' - ',num2str(opt.c2),'*(ux_x  + uy_y) = 0'];

% Set homogenous Dirichlet boundary coefficients.
   fea.phys.poi.bdr.coef{1,end}  = repmat({0},1,n_bdr);
   fea.phys.poi2.bdr.coef{1,end} = repmat({0},1,n_bdr);

% Check and parse physics modes.
   fea = parsephys(fea);

 else

   fea.dvar  = { 'u' 'v' };              % Dependent variable names.
   fea.sfun  = { opt.sfun opt.sfun };    % Shape functions.

% Mass matrix.
   fea.eqn.form = { [1;1] []    ;
                    []    [1;1] };
   fea.eqn.coef = { 1  [] ;
                    [] 1  };

% System/iteration matrix A_u := 0, A_vu = c2*( u_xx + u_yy ).
```

```
    if( opt.iexpl==1 )
      a12 = [];
    else
      a12 = [1;1];
    end
    fea.eqn.a.form = { []        a12 ;
                       [2 3;2 3] [] };
    fea.eqn.a.coef = { []     -1 ;
                       opt.c2 [] };

% Source term  f_u := v, f_v := 0.
    fea.eqn.f.form = {  1 ; 1 };
    if( opt.iexpl==1 )
      fea.eqn.f.coef = { 'v'; 0 };
    else
      fea.eqn.f.coef = { 0 ; 0 };
    end

% Define homogenous Dirichlet conditions.
    fea.bdr.d     = cell(2,n_bdr);
   [fea.bdr.d{:}] = deal( 0 );       % Zero Dirichlet conditions everyhere.
    fea.bdr.n     = cell(2,n_bdr);   % Clear Neumann conditions.

 end


% Parse and solve problem.
 fea = parseprob(fea);
 [fea.sol.u,fea.sol.tlist] = solvetime( fea, 'fid', fid, ...
                                        'init',  { opt.init 0 }, ...
                                        'icub',    4, ...
                                        'imass',   4, ...
                                        'tmax',    opt.tmax, ...
                                        'tstep',   opt.tstep, ...
                                        'ischeme', opt.ischeme );

% Postprocessing.
 xp = [0; 0];
 if ( opt.iplot>0 )
    figure
    subplot(1,2,1)
    isol = numel(fea.sol.tlist);
    postplot( fea, 'surfexpr', 'u', 'axequal', 'on', 'solnum', isol )
    title(['Solution at time ',num2str(fea.sol.tlist(isol))])

    for isol=1:numel(fea.sol.tlist)
      u(isol) = evalexpr( 'u', xp, fea, isol );
    end
    subplot(1,2,2)
    plot( fea.sol.tlist, u )
    title(['Solution at point (',num2str(xp'),')'])
    ylabel('u')
```

```
  xlabel('time')
end

u_ref    = -0.958;
[~,isol] = min(abs(fea.sol.tlist-1));
out.err  = abs( u_ref - evalexpr( 'u', xp, fea, isol ) )/abs(u_ref);
out.pass = out.err < opt.tol;
if ( nargout==0 )
  clear fea out
end
```

11.166 featool

FEATOOL Start the FEATool Multiphysics GUI.

11.167 fenics/fenics

FENICS Exports, solves, and imports a FEniCS project.

[PROB] =

FENICS(PROB, VARARGIN) Export, solves, or imports the solved problem described in the PROB finite element struct using the FEniCS project solver. Accepts the following property/value pairs.

```
Input       Value/{Default}              Description
-------------------------------------------------------------------------------
mode        data/export/solve/import     Command mode to call
data        default                      FEniCS input data file to use
fname       featool-fenics               FEniCS base filename root
fdir        string  {pwd}                Directory to write mesh and data files
order       scalar  {2}                  Integration order used in c-expressions
scmd        system default               Custom system solve command string
wbash       C:\Windows\System32\bash.exe Windows bash executable path
clear       boolean {true}               Clear output and log files
fid         scalar  {1}                  File identifier for output ([]=no output)
```

11.168 fenics/fenics_gui

FENICS_GUI Fenics project gui control dialog box.

11.169 openfoam/openfoam

OPENFOAM Exports, solves, and imports an OpenFOAM CFD problem.

[PROB] =

OPENFOAM(PROB, VARARGIN) Export, solves, or imports the solved problem described in the PROB finite element struct using the OpenFOAM CFD solver. Accepts the following property/value pairs, moreover also accepts the openfoam_data property/value pairs to set the OpenFOAM dicts during export.

```
Input       Value/{Default}              Description
-------------------------------------------------------------------------------
modes       check, export, solve, import Command mode(s) to call
data        default                      Default OpenFOAM data and parameter dict
interp      true                         Interpolate solution to grid points
casedir     default                      OpenFOAM case directory
foamdir     default                      OpenFOAM installation directory
logfname    featool-openfoam             OpenFOAM log/output filename
logfid      1                            Log file/message output file handle
tolres      1e-5                         Stopping criteria for initial residuals
clear       boolean {true}               Clear output and log files
```

See also

openfoam_data

11.170 openfoam/openfoam_data

OPENFOAM_DATA OpenFOAM control and solver parameters.

[DICT] =

OPENFOAM_DATA(VARARGIN) OpenFOAM control and solver parameter dict struct. Accepts the following property/value pairs.

```
Property      Value/{Default}         Description
```

```
-------------------------------------------------------------------------------
application    string {simpleFoam}    OpenFOAM application binary to run
ddtScheme      string {steadyState}   Time stepping scheme
tolres         scalar/vector {1e-5}   Stopping criteria for residuals (Simple)
startTime      scalar {0.0}           Simulation start time
endTime        scalar {100}           Simulation end time
deltaT         scalar {1.0}           Time step size
maxDeltaT      scalar {0.1}           Maximum time step size
maxCo          scalar {0.5}           Maximum Courant number
writeInterval  scalar {100}           Solution output write interval
purgeWrite     scalar {0}             Specified number of output solutions
transportModel string {Newtonian}     Transport model
simulationType string {laminar}       Simulation type laminar/RAS/LES
RASModel       string {kEpsilon}      RAS turbulence model
LESModel       string {Smagorinsky}   LES turbulence model
nu             scalar {1.0}           Kinematic viscosity (constant)
```

In addition to the specified property value pairs. Dict properties can be set and defined by specifying a cell array with value.

For example

```
    openfoam_data( {<tt>system</tt>,<tt>fvSolution</tt>,<tt>solvers</tt>,<tt>p</tt>,<tt>solver
```

sets the dict.system.fvSolution.solvers.p.solver field to GAMG. Furthermore, a n by 2 cell array value, such as for system.fvSchemes.divSchemes, can be used to assign several properties to a corresponding dict field.

See also

openfoam

11.171 openfoam/openfoam_gui

OPENFOAM_GUI OpenFOAM gui control dialog box.

11.172 openfoam/openfoam_write

OPENFOAM_WRITE Write OpenFOAM dict struct to file.

OPENFOAM_WRITE(DICT, FILE_NAME, FID_LOG) Write OpenFOAM dict struct to file.

11.173 geom/compute_distance

compute_distance Evaluate distance function.

[DIST] =

COMPUTE_DISTANCE(GOBJ, P_EVAL) Computes the distance function from geometry object GOBJ for evaluation points P_EVAL. GOBJ can be a valid geometry object, distance function handle, or anonymous function.

11.174 geom/extract_entries

EXTRACT_ENTRIES Extract array entries and recompute pointers.

[ARR_OUT, PTR_OUT] =

EXTRACT_ENTRIES(ARR_IN, PTR_IN, ORIENTATION, REM_DUPL) Extracts the entries (rows if ORIENTATION=1 or columns if ORIENTATION=2) from the array ARR_IN referenced in PTR_IN and returns the corresponding entries in ARR_OUT with recomputed local pointers PTR_OUT so that ARR_OUT(PTR_OUT) = ARR_IN(PTR_IN).

11.175 geom/geom_add_gobj

GEOM_ADD_GOBJ Add geometry object to geom or fea struct.

11.176 geom/geom_boundary_edges

GEOM_BOUNDARY_EDGES Constructs boundary edges from faces.

[GOBJ] =

GEOM_BOUNDARY_EDGES(GOBJ) Constructs boundary edges from the faces. Effectively adds an edge field to the boundaries struct with a n_e x 2 array of vertex pointers (pointing to the boundaries.vertices field).

11.177 geom/geom_combine_bbox

GEOM_COMBINE_BBOX Combine bounding boxes.

11.178 geom/geom_decompose_boundaries2

GEOM_DECOMPOSE_BOUNDARIES2 Decomposes intersecting boundaries of geometry objects.

[BOUNDARIES] =

GEOM_DECOMPOSE_BOUNDARIES2(ARGIN, SELECT) Decomposes intersecting boundaries of geometry objects. ARGIN may be a problem or geometry struct, or a cell array of geometry objects. The optional array SELECT indicates which objects should have their boundaries decomposed (the default is all boundaries).

11.179 geom/geom_decompose_boundaries3

GEOM_DECOMPOSE_BOUNDARIES3 Decomposes intersecting boundaries of geometry objects.

BOUNDARIES = GEOM_DECOMPOSE_BOUNDARIES3(C_OBJECTS, OP, SPLIT_ALL) Decomposes intersecting boundaries of two geometry objects in the cell array C_OBJECTS. OP is a string character (+, -, or &) representing the operation to perform (C_OBJECTS{1} OP C_OBJECTS{2}), and SPLIT_ALL is a flag to split all surface facets when checking for removal (default false).

11.180 geom/geom_get_tags

GEOM_GET_TAGS Get geometry object tags.

11.181 geom/geom_intersect_lines

GEOM_INTERSECT_LINES Compute intersection between two lines.

[S, T, TF] =

GEOM_INTERSECT_LINES(P_L1, P_L2) Computes the intersection point of two lines defined by the points P_L1 =
[x1 x2 y1 y2] and P_L2. Outputs are S and T which specify the relative position of the intersection points for each line, so that for example xi = x1+s∗(x2-x1). TF is a boolean which indicates if

the lines intersect within the segments defined by P1 (0<s<1) and P2 (0<t<1). Parallel non-intersecting lines return S and T = -inf, and TF = 0.

11.182 geom/geom_parse_formula

GEOM_PARSE_FORMULA Parse geom formula.

[TAGS, OPS] =

GEOM_PARSE_FORMULA(S) Parse geometry formula and return object tags and operations. Operators should be applied in sequence to the following geometry objects.

11.183 geom/geom_restrict_bbox

GEOM_RESTRICT_BBOX Restict bounding box.

[BBOX] =

GEOM_RESTRICT_BBOX(BBOX, GOBJ, N_POINTS) Restict bounding box around zero contour of GOBJ's distance function.

11.184 geom/geom_split_lines

geom_split_lines Split intersecting lines.

[P1, P2, TF] =

LINE_SPLIT(P_L1, P_L2) Tests if the line segments defined by P_L1 and P_L2 intersect and returns the corresponding coordinate(s) of splitting points in P1 and P2. TF indicates if and how the line segments should be split (with a negative value signifying

collinear lines)

```
  3: Split both line segments at P1 = P2
  2: Split line 2 at coordinate P2 (end point of line 1 touches line 2)
  1: Split line 1 at coordinate P1 (end point of line 2 touches line 1)
  0: Non-intersecting lines (P1 = P2 = [])
 -1: Split line 1 at P1 (and optionally P2), and remove line segment 2
 -2: Split line 2 at P2 (and optionally P1), and remove line segment 1
 -3: Identical lines detected (P1 = P2 = [])
```

See also

geom_intersect_lines

11.185 geom/geom_split_object

GEOM_SPLIT_OBJECT Split composite geometry object.

11.186 geom/geom_uniqify_tag

GEOM_UNIQIFY_TAG Uniquify geometry object tag.

11.187 geom/geom_update_tags

GEOM_UPDATE_TAGS Update the geometry tags field.

11.188 geom/gobj_boundary_vertbbox

GOBJ_BOUNDARY_VERTBBOX Compute vertices and bbox from boundaries.

[GOBJ] =

GOBJ_BOUNDARY_VERTBBOX(GOBJ) Use boundary information to compute gobj vertices and bbox.

11.189 geom/gobj_reorient_boundaries

GOBJ_REORIENT_BOUNDARIES Compute vertices and bbox from boundaries.

[GOBJ] =

GOBJ_REORIENT_BOUNDARIES(GOBJ, OFFSET) Use boundary information in GOBJ to re-orient boundaries. For edges tests wether the boundary mid point shifted by OFFSET (default 1e-6) in the normal direction is inside the geometry.

11.190 geom/order_boundaries

ORDER_BOUNDARIES Determine connected boundaries in order.

[BDR_ORDER] =

ORDER_BOUNDARIES(GEOM) Determines groups of connected boundary segments in two-dimensions. Returns a cell array BDR_ORDER for each geometry object with groups of connected boundary indexes. A negative index indicates that the orientation of the boundary is reversed with respect to boundary edges.

11.191 geom/plotgeom1

PLOTGEOM1 Visualize 1d geometries.

[H] =

PLOTGEOM1(SIN, VARARGIN) Function to plot and visualize geometry objects in one dimension. SIN is a valid fea problem struct or cell array of geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}           Description
-------------------------------------------------------------------------------
facecolor   [.8 .8 1]                 Face style or fixed color
edgecolor   {k}                       Edge line color
linestyle   {-}                       Edge line style
linewidth   {2}                       Edge line width
highlight   {}                        Highlight selected geometry objects
labels      off|{on}                  Show geometry labels
fontsize    {2*axes default}          Font size used in labels
axequal     off|{on}                  Axis equal setting
bbox        {0.05}                    Size of bounding box (0=off)
ngrid       {25}                      Background grid resolution
parent      {gca}                     Plot axes handle
```

See also

plotgeom

11.192 geom/plotgeom2

PLOTGEOM2 Visualize 2d geometries.

[H] =

PLOTGEOM2(SIN, VARARGIN) Function to plot and visualize geometry objects in two dimensions. SIN is a valid fea problem struct or cell array of geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}           Description
-------------------------------------------------------------------------------
facecolor   [.8 .8 1]                 Face style or fixed color
edgecolor   {k}                       Edge line color
linestyle   {-}                       Edge line style
linewidth   {1}                       Edge line width
highlight   {}                        Highlight selected geometry objects
labels      off|{on}                  Show geometry labels
fontsize    {2*axes default}          Font size used in labels
axequal     off|{on}                  Axis equal setting
bbox        {0.05}                    Size of bounding box (0=off)
ngrid       {25}                      Background grid resolution
parent      {gca}                     Plot axes handle
```

See also

plotgeom, plotgeomb

11.193 geom/plotgeom3

PLOTGEOM3 Visualize 3d geometries.

[H] =

PLOTGEOM3(SIN, VARARGIN) Function to plot and visualize geometry objects in three dimensions. SIN is a valid fea problem struct or cell array of geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}           Description
-------------------------------------------------------------------------------
facecolor   [.8 .8 1]                 Face style or fixed color
edgecolor   {k}                       Edge line color
linestyle   {-}                       Edge line style
linewidth   {1}                       Edge line width
highlight   {}                        Highlight selected geometry objects
labels      off|{on}                  Show geometry labels
fontsize    {2*axes default}          Font size used in labels
axequal     off|{on}                  Axis equal setting
bbox        {0.05}                    Size of bounding box (0=off)
```

```
ngrid       {25}                    Background grid resolution
alpha       {0.5}                   Transparency level
parent      {gca}                   Plot axes handle
```

See also

plotgeom

11.194 geom/plotgeomb

PLOTGEOMB Visualization of geometry boundaries.

[H] =

PLOTGEOMB(SIN, VARARGIN) Function to plot and visualize geometry object boundaries. SIN is a valid fea problem struct or cell array of geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}          Description
-------------------------------------------------------------------------------
axis        off|{on}                 Show axes
grid        on |{off}                Show grid
axequal     off|{on}                 Axis equal setting
linestyle   {-}                      Edge line style
linewidth   {1}                      Edge line width
arrowsize   {-0.025}                 Size of arrows (negative indicates relative)
colors      {r,g,b,c,m,y}            Cell array with colors
labels      on/{off}                 Print boundary numbers
fontsize    {2*axes default}         Font size used in text labels
selbdr      {all}                    Index vector to boundaries to plot
plotint     {0}                      Plot internal boundaries
merge       {1}                      Flag to merge geometry objects
parent      {gca}                    Axes handle to plot in
```

See also

plotgeom, plotbdr, plotsubd, plotgrid, postplot

11.195 geom/plotpoint

PLOTPOINT Visualization of point geometry object.

11.196 geom/polyarea3

POLYAREA3 Area of a polygon in 3D space.

[AREA] =

POLYAREA3(P1, P2, P3, P4) Computes the area of a triangular or quadrilateral 3D polygon. P1 is either a 3 (x/y/z) by 3 or 4 array of polygon vertex coordinates, or alternatively P1-P3/4 are arrays of size 3 by the number of polygons with coordinates of the polygon vertices.

11.197 geom/reconstruct_boundaries

RECONSTRUCT_BOUNDARIES Reconstruct boundaries.

[B] =

RECONSTRUCT_BOUNDARIES(P, E, F, V) Reconstructs boundaries from decomposed geometry information.

11.198 geom/reconstruct_triangles

RECONSTRUCT_TRIANGLES Reconstruct triangles from point cloud.

T = RECONSTRUCT_TRIANGLES(P, TOL) Reconstruct triangles from points in P by projecting points to the plane of the triangle formed by the first three points and then using 2D Delaunay triangulation. TOL gives the minimum acceptable triangle area.

11.199 geom2/bbox_intersect

BBOX_INTERSECT Compute and test if bounding boxes intersect.

[TF] =

BBOX_INTERSECT(P1, P2, COMPUTE_BBOX, TOL) Tests if the bounding boxes with coordinates P1 and P2 intersect. If the COMPUTE_BBOX is true the bounding boxes will be determined from P1 and P2. TOL specifies a distance tolerance (default eps).

11.200 geom2/boundary_retriangulation

BOUNDARY_RETRIANGULATION Retriangulate 3D boundaries.

[GOBJ] =

BOUNDARY_RETRIANGULATION(GOBJ, MODE, IS_WARN) Retriangulates boundaries in the 3D geometry object GOBJ. MODE (default 0) specifies if all edges should be checked for intersecting points for other boundaries.

11.201 geom2/compress_boundary_edges

COMPRESS_BOUNDARY_EDGES Compress boundary edges.

[VERTICES, EDGES, IND_E] =

COMPRESS_BOUNDARY_EDGES(BOUNDARIES) Compress edges in the BOUNDARIES struct and return an array of unique vertex coordinates (VERTICES) and a n_e by 2 array of edge connectivities in EDGES. The optional output IND_E is a vector with numbering for edge groups (grouping by boundary intersections).

11.202 geom2/convert_gobj_polygons

CONVERT_GOBJ_POLYGONS Constructs polygons from geometry object.

[POLYGONS] =

CONVERT_GOBJ_POLYGONS(GOBJ, ID) Constructs polygons from the boundaries of geometry object GOBJ by calling the CSG_OP build operation. ID if present will be appended to the polygon id property. Thus the final polygon identity field will consist of the local boundary number, local polygon number, and ID.

11.203 geom2/csg_op

CSG_OP Perform CSG operations.

[C, (D), STAT] =

CSG_OP(A, B, OP, DO_MERGE, TOL) Applies CSG operation OP to trees A and B, where OP is a string character indicating operation to perform (+ union/join, − subtract, & intersect). If DO_MERGE is true (default) the resulting CSG trees are merged, pruned, and returned in C, or if two outputs are requested the merged CSG trees from A and B are returned in C and D, respectively. The status return code STAT is zero if the operation has been applied sucessfully, and a positive integer to indicate error in which case the original inputs are returned unmodified.

With OP equal to b then the CSG build operation will be called on the polygons or vertices in A (with optional polygon identity argument in B). OP e extracts and returns all polygons from the CSG tree A (and corresponding CSG node normals in D). OP u updates the polygon vertices with those in B, assuming same order as the extract operation. If only two input arguments are given, then B will be used as OP. OP i will add the contents of B to the polygon identities. OP p will prune the CSG tree A, that is remove empty CSG nodes. OP v and w plots and visualizes the polygons and CSG tree nodes in A, respectively.

TOL (default 1e-7) specifies the tolerance for determining point

relative to splitting plane positions

```
t =  dot(plane_normal,point) - dot(plane_normal,point_in_plane)
```

so that t>tol is in front of the plane, and t<-tol is behind.

The CSG tree data structure are nested 1 x 5 cell arrays with the

following contents for each node

{ single point coordinates in node plane, plane normal, nested front nodes, nested back nodes, polygons }

The polygon data structure is a 6 x n_polygons cell array where the respective rows correspond to: polygon indices, number of vertices per cell, normal, vertex coordinates, bounding circle/sphere center, and radius.

11.204 geom2/csg_polygon_recombination

CSG_POLYGON_RECOMBINATION Recombine and tesselate polygons.

[POLYGONS] =

CSG_POLYGON_RECOMBINATION(POLYGONS, IS_WARN, TOL) Recombine co-linear/planar line segments and polygons. The IS_WARN flag (default false) indicates is polygon recombination is skipped due to differing identities.

11.205 geom2/csg_polygon_tesselation

CSG_POLYGON_TESSELATION Recombine and tesselate polygons.

[VERTICES, FACES, IDS] =

CSG_POLYGON_TESSELATION(POLYGONS, TOL) Recombine and tesselate POLYGONS.

11.206 geom2/geom_analyze

GEOM_ANALYZE Analyze geometry objects.

[SOUT] =

GEOM_ANALYZE(SIN, TOL, IS_WARN) Analyzes geometry (objects) SIN and returns a decomposed geometry with non-overlapping geometry objects. TOL (default 1e-5) specifies the tolerance for CSG boundary operations. The IS_WARN flag (default false) enables throwing warnings if the geometry objects cannot be combined.

11.207 geom2/geom_apply_formula

GEOM_APPLY_FORMULA Apply formula to a geometry struct.

[SOUT, STAT] =

GEOM_APPLY_FORMULA(SIN, FORMULA, TOL, IS_WARN) Applies a FORMULA to geometry objects in the fea or geometry struct SIN. The FORMULA is a string containing combinations of geometry object tags and operations (+ union/join, − subtract, & intersect). TOL (default 1e-5) specifies the tolerance for CSG boundary operations. The status return code STAT is zero if the operation has been applied sucessfully, or a positive integer corresponding to the number of failed geometry object combinations. The IS_WARN flag (default false) enables throwing warnings if the geometry objects cannot be combined.

11.208 geom2/geom_apply_transformation

GEOM_APPLY_TRANSFORMATION Apply transformation to geometry objects.

[SOUT] =

GEOM_APPLY_TRANSFORMATION(SIN, TAGS, DP, S, TH, AX) Applies transformations to the geometry objects with specified TAGS. DP is a 1 x n_sdim array specifying linear translations in

the coordinate directions. S applies a scaling (1 x n_sdim), and TH specifies a rotation in radians around the origin in 2D, and AX-axis (1=x default, 2=y, 3=z) in 3D.

11.209 geom2/geom_combine_objects

GEOM_COMBINE_OBJECTS Combine geometry objects.

[GEOM, NEW_TAG, STAT] =

GEOM_COMBINE_OBJECTS(GEOM, TAG1, TAG2, OP, TOL, IS_WARN) Combines geometry objects with TAG1 and TAG2 through application of OP, where OP is a string character indicating operation to perform (+ union/join, − subtract, & intersect). TOL (default 1e-5) specifies the tolerance for CSG boundary operations. The status return code STAT is zero if the operation has been applied sucessfully, and a positive integer to indicate error. The IS_WARN flag (default false) enables throwing warnings if the geometry objects cannot be combined.

11.210 geom2/geom_decompose

GEOM_DECOMPOSE Analyze and decompose geometry.

[P,E,F,V,IP] =

GEOM_DECOMPOSE(GEOM, TOL, IS_ANALYZED) Decomposes the geometry struct GEOM into unique points/vertices P, edges/line segments E, faces or surfaces F, and volumes V (for 3D geometries). IS_ANALYZED (default false) indicates if the geometry has already been processed by GEOM_ANALYZE. The optional input argument TOL defines the tolerance for deduplication and zeroing vertex coordinates (default eps∗1e3). IP indicates which vertices in P are due to point geometry objects (and not part of edges, faces, or volumes).

Geometries consisting of multiple geometry objects will be analyzed and if required split into separate non-overlapping objects. The decomposition process tries to apply one intersection and two subtraction operations to each combination of two geometry separate objects.

The output argument P is a array of unique points/vertices of size (n_p,n_sdim).

The two last columns of the edge/line segment array E (size (n_e,3)) contains start and end indices to vertices in P that make up the edges. The first column indicates edge segment groupings (corresponding to boundaries in 2D and geometry edges in 3D, No defined edge grouping is indicated with zeros in 3D).

The face/surface array F of size (n_v,2+n_max_e) consists of indices to edges that make up the faces (a negative entry indicates reversed edge direction). Similar to the first column of E,

the first column of F also indicates face groupings (integer > 0). The second boolean column indicates if each face is an outer boundary (0) or a hole (1). For faces with less than n_max_edges (the maximum number edges in any face) the extra entries are padded with zeros.

The volume array V (for 3D) indicates which faces F constitutes the volumes (size n_v,2+n_max_f) where a negative entry also here indicates reversed face direction. Similarly to F, the first column specifies the volume group, and the second if the entry volume describes the outer (external) or inner volume boundary (hole).

See also

gobj_decompose, geom_analyze

11.211 geom2/gobj_apply_op

GOBJ_APPLY_OP Apply operation to geometry objects.

[GOBJ, STAT] =

GOBJ_APPLY_OP(GOBJ1, GOBJ2, OP, TOL, IS_WARN) Applies operation OP to geometry objects GOBJ1 and GOBJ2, where OP is a string character indicating operation to perform (+ union/ join, − subtract, & intersect). The resulting resulting geometry object(s) is/are returned in a GOBJ struct array. TOL (default 1e-5) specifies the tolerance for CSG boundary operations. The status return code STAT is zero if the operation has been applied sucessfully, and a positive integer to indicate error. The IS_WARN flag (default false) enables throwing warnings if the geometry objects cannot be combined.

11.212 geom2/gobj_block

GOBJ_BLOCK Create block geometry object.

[GOBJ] =

GOBJ_BLOCK(XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX, TAG, T) Creates a block geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}          Description
-------------------------------------------------------------------------------
xmin        scalar  {0}              Minimum x-coordinate
xmax        scalar  {1}              Maximum x-coordinate
ymin        scalar  {0}              Minimum y-coordinate
ymax        scalar  {1}              Maximum y-coordinate
```

```
zmin        scalar  {0}             Minimum z-coordinate
zmax        scalar  {1}             Maximum z-coordinate
tag         string  {B1}            Geometry object tag/name
t           logical {false}         Triangulate boundary segments
```

11.213 geom2/gobj_circle

GOBJ_CIRCLE Create circle geometry object.

[GOBJ] =

GOBJ_CIRCLE(P, R, TAG) Creates a circle geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}         Description
-------------------------------------------------------------------------------
p           array  {[0 0]}          Coordinates of center point
r           scalar {1}              Circle radius
tag         string {C1}             Geometry object tag/name
```

11.214 geom2/gobj_cylinder

GOBJ_CYLINDER Create cylinder geometry object.

[GOBJ] =

GOBJ_CYLINDER(P, R, L, AX, TAG, N_S, T) Creates a cylinder geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}         Description
-------------------------------------------------------------------------------
p           array   {[0,0,0]}      Coordinates of base center point
r           scalar  {1}             Cylinder radius
l           scalar  {1}             Cylinder length
ax          scalar/array {1}        Axis direction (1/2/3 = x/y/z-axis)
                                    alt. axis direction vector (ex. [1,1,0])
tag         string  {C1}            Geometry object tag/name
n_s         scalar  {16}            Number of circumferential boundary segments
t           logical {false}         Triangulate boundary segments
```

11.215 geom2/gobj_decompose

GOBJ_DECOMPOSE Analyze and decompose geometry object.

[P,E,F,V] =

GOBJ_DECOMPOSE(GOBJ, TOL) Decomposes the geometry object GOBJ into unique points/vertices P, edges/line segments E, faces/surfaces F, and volumes V (for 3D objects). The optional input argument TOL defines the tolerance for vertex deduplication and zeroing vertex coordinates (default eps∗1e3).

The output argument P is a array of unique points/vertices of size (n_p,n_sdim).

The two last columns of the edge/line segment array E (size (n_e,3)) contains start and end indices to vertices in P that make up the edges. The first column indicates edge segment groupings (corresponding to boundaries in 2D and geometry edges in 3D). No defined edge grouping is indicated with zeros.

The face/surface array F of size (n_v,2+n_max_e) consists of indices to edges that make up the faces (a negative entry indicates reversed edge direction). Similar to the first column of E, the first column of F also indicates face groupings (integer > 0). The second boolean column indicates if each face is an outer boundary (0) or a hole (1). For faces with less than n_max_edges (the maximum number edges in any face) the extra entries are padded with zeros.

The volume array V (for 3D) indicates which faces F constitutes the volumes (size n_v,2+n_max_f) where a negative entry also here indicates reversed face direction. Similarly to F, the first column specifies the volume group, and the second if the entry volume describes the outer (external) or inner volume boundary (hole).

See also

geom_decompose

11.216 geom2/gobj_ellipse

GOBJ_ELLIPSE Create ellipse geometry object.

[GOBJ] =

GOBJ_ELLIPSE(P, RX, RY, TAG) Creates an ellipse geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}          Description
-------------------------------------------------------------------------------
```

```
p           array  {[0 0]}           Coordinates of center point
rx          scalar {1}                Radius along x-axis
ry          scalar {0.5}              Radius along y-axis
tag         string {E1}               Geometry object tag/name
```

11.217 geom2/gobj_empty

GOBJ_EMPTY Create empty geometry object.

[GOBJ] =
GOBJ_EMPTY(V, TAG) An empty geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}           Description
-----------------------------------------------------------------------------
v           array {[0 0;1 1]}         Min/max coordinates (2 x n_sdim array)
tag         string {E1}               Geometry object tag/name
```

11.218 geom2/gobj_grid

GOBJ_GRID Create background grid for geometry objects.

11.219 geom2/gobj_line

GOBJ_LINE Create line geometry object.

[GOBJ] =
GOBJ_LINE(P, TAG) Creates a line geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}           Description
-----------------------------------------------------------------------------
p           array  {[0;1]}            Start and end line coordinates (2 x n_sdim)
tag         string {L1}               Geometry object tag/name
```

11.220 geom2/gobj_point

GOBJ_POINT Create point geometry object.

[GOBJ] =

GOBJ_POINT(P, TAG) Creates a point geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}           Description
-------------------------------------------------------------------------------
p           array  {[0 0]}            Coordinates of point
tag         string {P1}               Geometry object tag/name
```

11.221 geom2/gobj_polygon

GOBJ_POLYGON Create polygon geometry object.

[GOBJ] =

GOBJ_POLYGON(P, TAG) Creates a polygon geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}           Description
-------------------------------------------------------------------------------
p           array  {[0 0;1 0;0 1]}    Polygon vertex points
tag         string {P1}               Geometry object tag/name
```

11.222 geom2/gobj_rectangle

GOBJ_RECTANGLE Create rectangle geometry object.

[GOBJ] =

GOBJ_RECTANGLE(XMIN, XMAX, YMIN, YMAX, TAG) Creates a rectangle geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}           Description
-------------------------------------------------------------------------------
xmin        scalar {0}                Minimum x-coordinate
xmax        scalar {1}                Maximum x-coordinate
```

```
ymin        scalar {0}              Minimum y-coordinate
ymax        scalar {1}              Maximum y-coordinate
tag         string {R1}             Geometry object tag/name
```

11.223 geom2/gobj_sphere

GOBJ_SPHERE Create sphere geometry object.

[GOBJ] =

GOBJ_SPHERE(P, R, AX, TAG, N_S, T) Creates a sphere geometry object. Accepts the following input parameters.

```
Parameter   Value/{Default}          Description
-------------------------------------------------------------------------------
p           array   {[0 0 0]}        Coordinates of center point
r           scalar  {1}              Sphere radius
ax          scalar/array {1}         Axis direction (1/2/3 = x/y/z-axis)
                                     alt. axis direction vector (ex. [1,1,0])
tag         string  {S1}             Geometry object tag/name
n_s         scalar  {16}             Number of circumferential boundary segments
t           logical {false}          Triangulate boundary segments
```

11.224 geom2/match_rows

MATCH_ROWS Find rows with identical content in A.

[IND_MATCH] =

MATCH_ROWS(A) Sorts indivitual rows in A and finds matches. Returns the parewise matching row indices in IND_MATCH (one row per match, where the column entries specify the first and second matching rows in A).

11.225 grid/assign_bdr

ASSIGN_BDR Assign boundary numbering according to geometry.

[GRID] =

ASSIGN_BDR(GRID, GEOM, E) Renumbers the boundaries in GRID.B(3,:) according to the

boundaries specified in the GEOM struct. In 3D the input argument E specified edges which split the geometrical boundaries, or alternatively is a scalar indicating the quadrature rule with which to split the geometric boundary for matching (default 5).

11.226 grid/blockgrid

BLOCKGRID Generate 3D regular tensor product hexahedral grid.

[GRID] =

BLOCKGRID(N_CX, N_CY, N_CZ, XP) Generates a tensor product block grid with N_CX, N_CY, NCZ hexahedral cells in the x, y, and z-directions. The optional argument XP=
[x1 y1 z1; x2 y2 z2] specifies the coordinates of the lower left (x1,y1,z1) and upper right (x2,y2,z2) corners. A call without input arguments generates a default 10x10x10 unit cube.

Examples

1) A 10x10x10 grid on the domain spanned by[0..1, 0..1, 0..1]

```
grid = blockgrid();
```

2) A 20x40x80 grid on the domain spanned by[-0.5..0.5, 0..2, 0..1]

```
grid = blockgrid( 20, 40, 80, [-0.5 0.5; 1 2; 0 1] );
```

See also

circgrid, cylgrid, holegrid, linegrid, rectgrid, ringgrid, spheregrid

11.227 grid/constrain_edges

CONSTRAIN_EDGES Constrains edges in a triangulation.

[P, T, DO_RETRI] =

CONSTRAIN_EDGES(P, T, EC, N_SDIM, IND_EXCL) Constrains edges EC in a triangulation T with grid points P. DO_RETRI indicates if any grid points or cells have been removed and a retriangulation is required.

11.228 grid/circgrid

CIRCGRID Generate 2d quadrilateral grid for a circle.

[GRID] =

CIRCGRID(NS, NR, R, XP, TH_OFFSET) Generates a quadrilateral grid for a circular domain with NS+NR cells in the radial direction. NS specifies the cell resolution of the inner square (default 4), and NR the number of cells in the radial direction of the outer layer (default 3). The optional arguments R and XP =
[x0;y0] specify the radius and center coordinates of the circle (default R = 1 and XP =
[0;0]). Furthermore, TH_OFFSET specifies a rotation of the whole grid.

Examples

1) A 64 cell grid for a circle with radius 1 centered at[0, 0]

```
grid = circgrid();
```

2) A 1024 cell grid for a circle with radius 0.5 centered at[1, 1]

```
grid = circgrid( 16, 12, 0.5, [1;1], 0 );
```

See also

cylgrid, blockgrid, holegrid, linegrid, rectgrid, ringgrid, spheregrid

11.229 grid/cylgrid

CYLGRID Generate 3d cylindrical hexahedral grid.

[GRID] =

CYLGRID(NS, NR, NZ, R, LZ, XP, AX) Generates a hexahedral grid for a cylindrical domain with NS+NR cells in the radial direction. NS specifies the cell resolution of the inner square (default 4), and NR ns number of cells in the radial direction of the outer layer (default 3). NZ specity the numver of cells in the lengthwise direction and LZ the corresponding length. The optional arguments R, XP, and AX specify the radius, center coordinates of the cylinder (default R = 1 and XP =
[0;0;0]), and the axis of alignment (default 1 equals to the x-axis).

Examples

1) A cylindrical grid with radius 1 and length 1

```
grid = cylgrid();

2) Cylinder with radius 0.5 length 2 centered and extending
   in the negative y-direction from the point [1 1 2].

grid = cylgrid( 3, 4, 10, 0.5, 2, [1;1;2], -2 );
```

See also

blockgrid, circgrid, holegrid, linegrid, rectgrid, ringgrid, spheregrid

11.230 grid/delcells

DELCELLS Delete cells from grid.

[SOUT] =

DELCELLS(SIN, SELECTION) Deletes cells from a grid.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
sin         struct                 Grid or problem struct with
                                   p, c (and s) fields
selection   array                  Array with indicies or string expression
                                   indicating cells to delete
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
sout        struct                 Output grid or problem struct
```

11.231 grid/findbdr

FINDBDR Find boundary indices for an expression.

[BDR, INDB] =

FINDBDR(SIN, S_EXPR, MATCH_ALL) Evaluates S_EXPR for boundary nodes in the grid or fea struct and returns the boundary numbers BDR (corresponding to values in grid.b(3,:)) for which the expression is true, and also indices INDB to the corresponding boundary segments (columns in grid.b). The flag MATCH_ALL requires all nodes on a boundary to fulfill S_EXPR to return the found indices (default). If MATCH_ALL is false all indices to all boundaries BDR, edges, and faces INDB are returned.

Example

```
% Set boundary number 2 on all boundary edges/faces where x>0.5.
[~,indb] =
    findbdr( grid, x>0.5, false ); b(3,indb) = 2;
```

11.232 grid/gridadj

GRIDADJ Generate grid cell adjacency information.

[A] =

GRIDADJ(C, N_SDIM) Generates an array A with indices to neighboring cells and zeros where no neighbors are found. Input data is an array C with cell connectivities and N_SDIM specifying the number of space dimensions.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
c           (n_v,n_c)              Cell connectivity pointer array where the
                                   entries in each column correspond to the
                                   cell vertex numbers specifying the cells.
n_sdim      scalar                 Number of space dimensions.
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
a           (n_e,n_c)              Array with cell adjacency pointers where
                                   the entries in each column correspond to
                                   the neighbouring cell number for each
                                   edge or face. A zero entry indicates that
                                   the edge/face belongs to a boundary.
```

11.233 grid/gridalign2

GRIDALIGN2 Align vertices of quadrilateral cells with interface.

[P, C, U] =

GRIDALIGN2(P, C, DIST, VERT, HTOL, P_INTF) Aligns quadrilateral grid cell edges with the zero level of the distance function DIST. Accepts the following input and output arguments.

```
Property    Value/{Default}          Description
-------------------------------------------------------------------------------
p           array  [n_p,2]           Array of grid vertices
c           array  [n_c,4]           Array of quadrilateral connectivities
dist        anonymous function       Distance function or geometry object
vert        array   [n_pfix,2]       Points that must be in the grid
htol        scalar  {0.2}            Cell size width fraction in grid snapping
p_intf      array  [2*n_l+nan,2]     Interface line segemet array (line format)
u           array  [n_p,1]           Output array of distance values in vertices
```

See also

gridgen_quad, quadmesh

11.234 grid/gridbdr

GRIDBDR Generate grid boundary information.

[B] =

GRIDBDR(P, C, A, THLIM, FIX_BDR, B0) Given arrays P, C and A with cell connectivities and adjacency information generates an array with boundary information. The last three input arguments are optional.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
p           (n_sdim,n_p)           Coordinate array for grid vertices.
c           (n_pcell,n_cells)      Cell connectivity pointer array where the
                                   entries in each column correspond to the
                                   cell vertex numbers.
a           (n_pcell,n_cells)      Array with cell adjacency pointers where
                                   the entries in each column correspond to
                                   the neighbouring cell number for each edge
                                   or face. A zero entry indicates that the
                                   edge/face belongs to a boundary.
thlim       scalar {0.77}          boundary edge detection limit, splits boundaries
                                   with cos(normal angles) less than thlim.
fix_bdr     logical {0}            collect all boundary edges to the same boundary
b0          (4/5/6,n_b0)           Array of partially precomputed boundaries
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
b           (4/5/6,n_b)            Array with boundary information, the first row
                                   indicates the cell number, followed by cell
                                   edge/face, boundary number, and the components
                                   of the normal vector for each boundary edge/face
```

See also

gridbdr1

11.235 grid/gridbdr1

GRIDBDR1 Generate grid boundary information for one dimensional cells.

[B] =

GRIDBDR1(P, C, A) Given arrays P, C and A with cell connectivities and adjacency information generates an array with boundary information.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
p           (1,n_p)                Coordinate array for grid vertices
```

```
c           (2,n_cells)           Cell connectivity pointer array where the
                                  entries in each column correspond to the
                                  cell vertex numbers.
a           (2,n_cells)           Array with cell adjacency pointers where
                                  the entries in each column correspond to
                                  the neighbouring cell number for each
                                  edge. A zero entry indicates that the edge
                                  belongs to a boundary.
                                                                              .
Output      Value/(Size)          Description
-------------------------------------------------------------------------------
b           (4,n_b)               Array with boundary information, first row
                                  gives the cell number, second the cell edge
                                  number, the third the boundary number, and
                                  the last row the outward normal.
```

See also

gridbdr

11.236 grid/gridbdrx

GRIDBDRX Reconstruct interior boundaries.

[GRID] =
GRIDBDR(GRID) Reconstructs interior boundaries and updates the GRID.B field.

See also

gridbdr

11.237 grid/gridbfstf

GRIDBFSTF Find starting face.

[IX] =
GRIDBFSTF(CELLS, FACES, LOCAL_VERTICES_PER_FACE) Finds edge or face connected to the least amount of vertices. CELLS is an array of vertex connectivities corresponding to each face in FACES. FACES is a vector of face numbers for belonging to a boundary. LOCAL_VERTICES_PER_FACE is an array with local vertex numbering per face. Output IX is a number corresponding to the found column in CELLS/FACES.

11.238 grid/gridcheck

GRIDCHECK Check grid for errors.

[N_ERR, CIND] =

GRIDCHECK(G, ISWARN, I_CHECK) Checks a grid G for errors. N_ERR returns the number of errors found. CIND is a cell array giving the indices to incorrectly oriented cells, collapsed cells (with zero volume/area), and non-convex cells. ISWARN is a flag to enable/disable warnings. I_CHECK is a flag to choose check/test type (1 = test Jacobian determinant (default), 2 = test if polygon points are in clockwise order, 3 = split hexahedra into six tetrahedra).

See also

reorient_cells

11.239 grid/gridconne

GRIDCONNE Group connected edges/faces.

[IND_GROUPS, N_GROUPS] =

GRIDCONNE(C, F, N_SDIM) Finds index to edges connected with vertices in 2D or faces connected with edges in 3D. C is an array of cell connectivities and F is a vector of n_c length inticating faces to check.

11.240 grid/gridedge

GRIDEDGE Generate grid cell edge numbering.

[E, EV] =

GRIDEDGE(C, N_SDIM) Generates an array E with global edge numbers. Input data is an array C with cell connectivities and N_SDIM specifying the number of space dimensions.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
c           (n_v,n_c)              Cell connectivity pointer array where the
                                   entries in each column correspond to the
                                   cell vertex numbers specifying the cells.
n_sdim      scalar                 Number of space dimensions.
                                                                              .
Output      Value/(Size)           Description
```

```
-------------------------------------------------------------------------------
e           (n_e,n_c)             Array with global edge numbers for all cells.
ev          (n_e,2)               Array with indices to vertices for each edge.
```

11.241 grid/gridextrude

GRIDEXTRUDE Extrude 2D grid to 3D.

[GRID] =

GRIDEXTRUDE(GRID, NZ, LZ, AX) Extrudes a 2D GRID to 3D. NZ is the number of cells in the direction of extrusion, and LZ the corresponding extrusion length. AX
[-3..3] optionally rotates and aligns the extruded grid with the selected axis (default 3 which is the z-axis).

Gridextrude supports both extruding 2D quadrilateral grids to 3D hexahedra, and unstructured simplex triangular grids to tetrahedra (via first extruding to prisms and then splitting each prism into three tetrahedra).

Examples

1) Extrude a ring to form a tube

```
grid = ringgrid( 2, 16 );
grid = gridextrude( grid, 10, 5, 3 );

2) Extrude a composite grid by a length of 3
   (in the positive x-direction, AX=1).

grid = holegrid( 6, 5 );
grid = delcells( grid, <tt>((x>0).*(y>0))==0</tt> );
grid = gridextrude( grid, 12, 3, 1 )

3) Generate unstructured circuit board grid.

ro = 0.003;   % Outer circle diameter.
ri = ro/2;    % Inner circle diameter.
```

% Circle 1. co1 = gobj_circle(
[0.01,0.01], ro, CO1); ci1 = gobj_circle(
[0.01,0.01], ri, CI1);

% Circle 2. co2 = gobj_circle(
[0.05,0.02], ro, CO2); ci2 = gobj_circle(
[0.05,0.02], ri, CI2);

% Remove inner holes from circles. geom.objects = { co1 ci1 co2 ci2 }; geom = geom_apply_formula(geom, CO1-CI1); geom = geom_apply_formula(geom, CO2-CI2);

% Base rectangle. r1 = gobj_rectangle(0, 0.06, 0, 0.03, R1);

% Remove (outer) circles from base rectangle. geom.objects =
[geom.objects { r1 co1 co2 }]; geom = geom_apply_formula(geom, R1-CO1-CO2);

% Polygon for connecting conductive track. w = 0.002/2; p =
[0.01 0.02-w/2 0.03-w/2 0.05 0.05 0.03+w/2 0.02+w/2 0.01 ; 0.01+w 0.01+w 0.02+w 0.02+w 0.02-w 0.02-w 0.01-w 0.01-w]; p1 = gobj_polygon(p , P1');

% Remove outer circles from conductive track. geom.objects =
[geom.objects { p1 co1 co2 }]; geom = geom_apply_formula(geom, P1-CO1-CO2);

% 2D grid generation. hmax = (ro-ri)/2; grid = gridgen(geom, hmax, hmax);

% Extrude grid. grid = gridextrude(grid, 2, 0.002);

See also

gridmerge, gridrevolve, gridrotate, gridscale

11.242 grid/gridface

GRIDFACE Generate grid cell face numbering.

[F, FV] =

GRIDFACE(C, B) Generates an array F with global face numbers and FV with grid point indices er face . Input data is an array C with cell connectivities, and optionally the boundary struct B if upper triangular (OpenFOAM) ordering should be used (default normal ordering).

Input	Value/(Size)	Description
c	(n_v,n_c)	Cell connectivity pointer array where the entries in each column correspond to the cell vertex numbers specifying the cells.
b	struct	Boundary struct to use upper triangular ordering

```
                                                                                .
Output      Value/(Size)           Description
--------------------------------------------------------------------------------
f           (n_f,n_c)              Array with global face numbers for all cells.
fv          (n_f,3/4)              Array with indices to vertices for each face.
```

11.243 grid/gridmerge

GRIDMERGE Merge two grids.

[GRID] =

GRIDMERGE(GRID1, IND1, GRID2, IND2, I_DELETE, I_CHECK) Merges the matching boundaries edges and faces specified by boundary numbers IND1 in GRID1 and IND2 in GRID2. The algorithm tries to match up the boundary with the fewest boundary edges/faces and nodes to the closest corresponding ones on GRID2. The optional argument I_DELETE selects which grid to delete duplicate nodes from (I_DELETE is 2 per default). I_CHECK toggles error checking on/off.

Examples

```
1) Create a 2D rivet by joining a rectangle and half circle.

grid1 = rectgrid( 4, 10, [0.3 0.7;0 1] );
grid2 = circgrid( 4, 3, 0.5, [0.5;1] );
grid2 = delcells( grid2, <tt>y<=1</tt> );
grid  = gridmerge( grid1, 3, grid2, 4 );

2) Create a flow over cylinder benchmark grid by merging three grids.

grid1 = ringgrid( [0.05 0.06 0.08 0.11 0.15], 32, [], [], [0.2;0.2] );
grid2 = holegrid( 8, 1, [0 0.41;0 0.41], 0.15, [0.2;0.2] );
grid2 = gridmerge( grid1, 5:8, grid2, 1:4 );
grid3 = rectgrid( [0.41 0.5 0.7 1 1.4 1.8 2.2], 8, [0.41 2.2;0 0.41] );
grid  = gridmerge( grid3, 4, grid2, 6 );

3) Create a 3D grid with two brackets attached to an I-beam section.

grid01 = ringgrid( 1, 20, 0.03, 0.06, [0;0] );
indc01 = selcells( grid01, <tt>y<=sqrt(eps)</tt> );
grid01 = delcells( grid01, indc01 );

grid02 = holegrid( 5, 1, .06*[-1 1;-1 1], .03, [0;0] );
indc02 = selcells( grid02, <tt>y>=-sqrt(eps)</tt> );
```

```
grid02 = delcells( grid02, indc02 );
grid2d = gridmerge( grid01, [5 6], grid02, [7 8] );

grid1 = gridextrude( grid2d, 1, 0.02 );
grid1 = gridrotate( grid1, pi/2, 1 );
grid2 = grid1;
grid1.p(2,:) = grid1.p(2,:) + 0.03;
grid2.p(2,:) = grid2.p(2,:) - 0.01;

x_coord = [ -0.08 linspace(-0.06,0.06,6) 0.08 ];
y_coord = [ -0.2 -0.15 -0.1 -0.05 -0.03 -0.01 ...
             0.01  0.03  0.05  0.1  0.15  0.2 ];
grid3 = blockgrid( x_coord, y_coord, 1, ...
                   [-0.08 0.08;-0.2 0.2;-0.08 -0.06] );
grid4 = blockgrid( 1, y_coord, 5, ...
                   [-0.01 0.01;-0.2 0.2;-0.18 -0.08] );
grid5 = grid3;
grid5.p(3,:) = grid5.p(3,:) - 0.12;

grid = gridmerge( grid1, 9, grid3, 6 );
grid = gridmerge( grid2, 9, grid, 17 );
grid = gridmerge( grid4, 6, grid, 23, 1 );
grid = gridmerge( grid5, 6, grid, 29, 2 );
```

See also

gridextrude, gridrevolve, gridrotate, gridscale

11.244 grid/gridnormals

GRIDNORMALS Compute normal vectors.

[N] =

GRIDNORMALS(P, C, B) Computes an array of normal vectors. P is an array of grid point coordinates and C grid vertex connectivities. B specify boundary edges/faces with the fist row B(1,:) indicating cell numbers and the second B(2,:) local cell edge/face numbering.

11.245 grid/gridrefine

GRIDREFINE Uniform refinement of a grid.

[SOUT] =
GRIDREFINE(SIN) Creates a uniform refinement of a grid.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
sin         struct                 Grid or problem struct with
                                   p, c (and a, s, and b) fields
fid         scalar/{1}             File identifier for output ([]=no output)
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
sout        struct                 Output grid or problem struct
```

See also

gridrefine1, gridrefine2, gridrefine3

11.246 grid/gridrefine1

GRIDREFINE1 Uniform refinement of a 1D grid.

[GRID] =
GRIDREFINE1(GRID) Creates a uniform refinement of a 1D grid.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
grid        struct                 Grid struct
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
grid        struct                 Refined grid struct
```

11.247 grid/gridrefine2

GRIDREFINE2 Uniform refinement of a 2D grid.

[GRID] =
GRIDREFINE2(GRID) Creates a uniform refinement of a 2D grid.

```
Input       Value/(Size)           Description
-----------------------------------------------------------------------------
grid        struct                 Grid struct
                                                                             .
Output      Value/(Size)           Description
-----------------------------------------------------------------------------
grid        struct                 Refined grid struct
```

11.248 grid/gridrefine3

GRIDREFINE3 Uniform refinement of a 3D grid.

[GRID] =

GRIDREFINE3(GRID) Creates a uniform refinement of a 3D grid.

```
Input       Value/(Size)           Description
-----------------------------------------------------------------------------
grid        struct                 Grid struct
do_mg       logical/{0}            Flag to use mg hexahedral cell orientation
                                                                             .
Output      Value/(Size)           Description
-----------------------------------------------------------------------------
grid        struct                 Refined grid struct
```

11.249 grid/gridrevolve

GRIDREVOLVE Revolve and extrude a 2D grid.

[GRID] =

GRIDREVOLVE(GRID, N_CZ, LEN, N_REV, R_OFF, TH_XZ, I_JOIN) Revolves and extrudes a 2D GRID to generate a 3D grid. N_CZ is the number of cells and LZ the length/height in the z-direction. N_REV specify the number of revolutions and R_OFF the radial offset of the original input grid. TH_XZ optionally prescribes an initial angle of the input grid plane. I_JOIN is a flag to join the two ends.

Examples

1) Create a torus grid

```
grid = gridrevolve( circgrid(), 20, 0, 1, 2, pi/2, 1 );
```

2) Create a spiral grid

```
grid = gridrevolve( rectgrid(3), 100, 10, 3, 2 );
```

2) Create a "cored apple"

```
grid = circgrid();
grid = delcells( grid, <tt>x>=0</tt> );
grid = gridrevolve( grid, 20, 0, 1, 1, pi/2, 1 );
```

See also

gridextrude, gridmerge, gridrotate, gridscale

11.250 grid/gridrotate

GRIDROTATE Rotate grid.

[GRID] =

GRIDROTATE(GRID, TH, AX) Applies rotation angle TH to grid points (in 3D around axis AX). ARGI can either be a grid struct or simply an array of coordinates. Normals will also be recalculated in the case of a grid struct with boundary information.

See also

gridextrude, gridmerge, gridrevolve, gridscale

11.251 grid/gridscale

GRIDSCALE Scale a grid.

[GRID] =

GRIDSCALE(GRID, S) Scaling a grid with the factors specified in S. S can either be a real array with constant scaling factors for each direction, for example S =
[1 2 1] for a 2 times scaling in the y-direction. Alternatively, S can be a cell array with string expressions for the scaling factor.

Examples

1) Scale a circle by a factor of 2 in the x-direction

```
grid = spheregrid();
grid = gridscale( grid, [2 1 1] );
```

2) Scale the upper half of a rectangle to form an angle

```
grid = rectgrid();
grid = gridscale( grid, {<tt>1-(y>0.5).*(y-0.5)</tt> 1} );
```

See also

gridextrude, gridmerge, gridrevolve, gridrotate

11.252 grid/gridsmooth

GRIDSMOOTH Grid smoothing.

[SOUT] =

GRIDSMOOTH(SIN, N_SM, ALPHA, I_SM) Apply grid to grid in SIN. N_SM specifies the number of smoothing steps to apply (default 3) and ALPHA is the relaxation parameter (default 0.8). I_SM specifies the smoothing method (default 2).

1: Laplacian smoothing operator defined as

```
      Pnew = (1-ALPHA)*P + ALPHA/#neighbours * SUM(Qj)
```

with Qj being the neighbours of the point P.

2: Density weighted umbrella operator defined as

```
 Pnew = (1-ALPHA) * P + ALPHA/SUM(|P-Qj|) * SUM(|P-Qj| * Qj)
```

with Qj being the neighbours of the point P. The constant ALPHA defines the weighting of the midpoint P of the "patch".

11.253 grid/gridvert

GRIDVERT Generate grid vertex adjacency map.

[V, VV] =

GRIDVERT(C, INDC, N_SDIM) Generates an sparse map V which indicates which cells are connected to each other through the nodes.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
c           (n_v,n_c)              Cell connectivity pointer array where the
                                   entries in each column correspond to the
                                   cell vertex numbers specifying the cells.
indc        vector                 Index to cells for which to compute map
                                   (all cells by default).
n_sdim      scalar                 Optional input argument indicating computation.
                                   of connectivities for connected edges.
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
v           sparse(n_c,n_c)        Sparse array indicating which grid cells are
                                   connected through the grid points. Or with
                                   three input arguments a which grid cells are
                                   connected though the edges (incl. n_sdim).
vv          sparse(n_p,n_p)        Sparse array which for each row indicates
                                   which nodes/edges are connected through cells.
```

11.254 grid/grid_boundary_nodes

GRID_BOUNDARY_NODES Get index to nodes on the boundaries.

[IND_BN] =

GRID_BOUNDARY_NODES(GRID, INTERNAL) Finds the index to nodes on the boundaries. The INTERNAL flag specifies if nodes on internal boundaries should be found or not (default).

11.255 impexp/geom2geo

GEOM2GEO Generate GEO geometry data for use with Gmsh.

[GEO] =

GEOM2GEO(SIN, HMAX, HMAXB, HMAXP, FILE_NAME, FID_LOG, TOL, USE_COMPOUND) Generate GEO data for use with the grid generator Gmsh.

SIN can either be a valid geometry or precomputed GEO struct, HMAX, and HMAXB are scalars or arrays indicating grid sizes for subdomains and boundaries. FILE_NAME is optional and enables output to file (writes a geo file, default featool_gmsh). FID_LOG specifies a message log file handle (negative for gui output or empty for no output). The optional input argument TOL defines the tolerance for deduplication and zeroing vertex coordinates (default eps∗1e3).

HMAX, HMAXB, and HMAXP are used to calculate and prescribe the mesh sizes. HMAX is a scalar or array prescribing mesh sizes for subdomains, while HMAXB and HMAXP corresponds to boundary and vertex mesh sizes. Mesh sizes are prescribed in Gmsh Restrict Fields for edges and faces, respectively. A non-zero and non-empty HMAX assigns mesh size values to boundary fields connected to subdomains. If HMAX is an array with positive values the minimum value will be used, while negative HMAX values prescribes the mean absolute for each boundary. If HMAXB is given the corresponding boundary values are overwritten (HMAXB overrides HMAX).

The USE_COMPOUND flag indicates if Gmsh compound lines and surfaces should be used (default true).

BIX output GEO struct field contains boundary numbers matching to physical followed by additional compound line entries in 2D and surfaces in 3D. BIX can thus be used to set geometrical boundary numbers during import.

11.256 impexp/geom2poly

GEOM2POLY Generate poly data for use with Triangle.

[POLY] =

GEOM2POLY(GEOM, HMAX, HMAX_E, FILE_NAME, FID_LOG) Generate poly data for use with the 2D unstructured grid generator Triangle. GEOM is a valid geometry struct, HMAX and HMAX_E are arrays or scalars indicating the maximum grid size for subdomains and edges, respectively. FILE_NAME is optional and enables output to file (writes a poly file). FID_LOG specifies a message log file handle (negative for gui output or empty for no output).

11.257 impexp/grid2foam

GRID2FOAM Generate/convert grid data to OpenFOAM format.

11.258 impexp/impexp_ascii

IMPEXP_ASCII Import or export data in ascii format.

[DATA] =

IMPEXP_ASCII(DATA_SPEC, DATA, FID_LOG) Import or export DATA according to DATA_SPEC in ASCII format. If DATA is missing or empty export is assumed. FID_LOG specifies a message log file handle (negative for gui output or empty for no output).

DATA_SPEC is a struct with the following optional fields

.pre - is a cell array of string commands to execute/evalutate before main processing.

.proc - is a cell array processed row by row for which the first column entry specifies a specific task/function and the following entires are arguments.

{`command` s_cmd} - Evaluates the string command S_CMD.

{`funcall` {s_vars} s_fun {args}} - Calls the function S_FUN with a cell array of input arguments ARGS, and a cell array of variable names for outputs in S_VARS.

{`goto_line` s_keyword} - Rewinds and scan file to find the line starting with S_KEYWORD. If S_KEYWORD is a cell STRCONV will be called to parse the line. In export/write mode the export it analogous to process_line.

{`process_block` s_var s_format siz} - Call FSCANF from current point with format S_FORMAT and optional argument SIZ. The output is assigned to the variable name in S_VAR. Writes block in ascii format in export mode.

{process_line {s_vars} s_format} - Reads line and calls STRCONV to convert into S_FORMAT (optional) and assigns result to variables in S_VARS. Writes variables in export mode.

{process_string {s_vars} s_format s_var} - Applies STRCONV to string variable S_VAR as above (instead of reading a new line).

{rewind} - Rewinds the file.

{string_line s} - Reads line to variable S_LINE import mode, or alternatively writes line in export mode.

OTHERWISE - The next line is read into the variable S_LINE or nothing in export mode.

.post - is a cell array of string commands to execute/evalutate after main processing.

11.259 impexp/impexp_dolfin

IMPEXP_DOLFIN Import/export grid in Dolfin XML format.

[GRID] =

IMPEXP_DOLFIN(FILE_NAME, MODE, DATA, USE_MESHFCN, FID_LOG) Import or export of a Dolfin/fenics (.xml) grid and data format. FILE_NAME is a string specifying the (root) file name to process. MODE can either be a string indicating import (no boundary reconstruction), import_bdr (boundary reconstruction with gridbdr), or export. For export, DATA can be either a full fea struct or just the grid struct. USE_MESHFCN is a boolean flag to write subdomain numbers in a mesh function instead of the (default) domain field. A GRID struct is output when importing. FID_LOG is an optional log file handle for message output (negative for gui output or empty for no output).

11.260 impexp/impexp_foam

IMPEXP_FOAM Import/export OpenFOAM format.

[U] =

IMPEXP_FOAM(DIRECTORY, MODE, PROB, FID_LOG) Import and export of OpenFOAM ASCII mesh and problem data. Returns the solution vector U (after import).

DIRECTORY specifies the location of OpenFOAM case files.

MODE can either be a string indicating IMPORT (default) or EXPORT.

PROB is a valid FEATool FEA problem struct.

DICT are user supplied OpenFOAM dict structs to write.

FID_LOG is an optional log file handle for message output (negative for GUI output or empty for no output).

11.261 impexp/impexp_gid

IMPEXP_GID Import/export grid in GID msh format.

[GRID] =

IMPEXP_GID(FILE_NAME, MODE, GRID, FID_LOG) Import or export of a GiD mesh ASCII format sh file. FILE_NAME is a string specifying the (root) file name to process. MODE can either be a string indicating import (no boundary reconstruction), import_bdr (boundary reconstruction with gridbdr), or export. GRID is a grid struct input for export and output for import. FID_LOG is an optional log file handle for message output (negative for gui output or empty for no output).

11.262 impexp/impexp_gmsh

IMPEXP_GMSH Import/export grid in GMSH format.

[GRID] =

IMPEXP_GMSH(FILE_NAME, MODE, DATA, IIND, FID_LOG, BIX) Import and export of Gmsh (sh) Ascii mesh and grid file format.

FILE_NAME is a string specifying the (root) file name to process.

MODE can either be a string indicating IMPORT (where boundary numbering are reconstructed from Gmsh physical or elemental tags if available), IMPORT_BDR (boundary reconstruction with gridbdr), or EXPORT.

With the IMPORT mode option the function returns a GRID struct. While for exporting, the input DATA can be given as either a valid fea finite element data struct, or a FEATool grid struct.

IIND is an optional array for physical domains (subdomains in 2D and boundaries in 3D) to extract (default all).

FID_LOG is an optional log file handle for message output (negative for Gui output or empty for no output).

BIX is an optional index vector to correct boundary numbering.

11.263 impexp/impexp_gmv

IMPEXP_GMV Import/export grid in GMV format.

[GRID] =

IMPEXP_GMV(FILE_NAME, MODE, DATA, N_SDIM, FID_LOG, SOLNUM) Import or export of a General Mesh Viewer (GMV) .gmv format. FILE_NAME is a string specifying the (root) file name to process. MODE can either be a string indicating import (no boundary reconstruction), import_bdr (boundary reconstruction with gridbdr), or export. For export, DATA can be either a whole fea struct or just the grid struct. N_SDIM is the number of space dimensions. And SOLNUM is an optional solution number to export. A GRID struct is output and returned after importing. FID_LOG is an optional log file handle for message output (negative for gui output or empty for no output).

11.264 impexp/impexp_stl

IMPEXP_STL Import/export geometry in STL format.

[GEOM] =

IMPEXP_STL(FILE_NAME, MODE, DATA, TOL, FID_LOG) Import and export of STL format CAD geometries. Both full 3D and 2D planar geometries are supported (3D planar STL data will be converted to 2D geometry objects).

FILE_NAME is a string specifying the (root) file name to process, or a cell array of strings specifying several STL files where each file will be parsed into its separate boundary during import.

MODE can either be a string indicating IMPORT or EXPORT. With the IMPORT mode option the function returns a GEOM struct. While for exporting, the input DATA can be given as either a valid fea finite element data struct, or a FEATool geometry struct.

TOL is a vector optional of tolerance settings during import. The first tolerance controls the deduplication of points (default 1e-3 relative tolerance). The second entry controls tolerance for grouping. The third entry is an integer scalar specifying the number of facets in groups to merge together (default 1 only merging ungrouped facets).

FID_LOG is an optional log file handle for message output (negative for Gui output or empty for no output).

11.265 impexp/impexp_triangle

IMPEXP_TRIANGLE Import/export grid data in Triangle format.

[DATA] =

IMPEXP_TRIANGLE(FILE_NAME, MODE, DATA, HMAX, HMAX_E, FID_LOG) Import or export of Triangle grid data in .node/.ele/.edge/.poly format. FILE_NAME is a string specifying the (root) file name to process. MODE can either be a string indicating import or export. For export, DATA can be either a whole fea struct or just the geom struct which is used to compute the poly information used by Triangle. HMAX and HMAX_E specifies grid sizes for subdomains and edges when exporting. A DATA grid struct is output when importng. FID_LOG is an optional log file for message output.

11.266 impexp/log_message

LOG_MESSAGE Output message to file, terminal, or gui.

11.267 impexp/open_file

OPEN_FILE Open file for reading and writing.

[FID, FILE_NAME] =

OPEN_FILE(FILE_NAME, FILE_EXTENSIONS, FID_LOG) Opens file FILE_NAME with optional FILE_EXTENSIONS (string or cell array of strings). FID_LOG optionally specifies a log file for messages.

11.268 grid/hex2tet

HEX2TET Convert hexahedra to tetrahedra.

[SOUT] =

HEX2TET(SIN, MODE) Convert hexahedral to tetrahedral cells.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
sin         struct                 Grid or problem struct with
                                   p, c (and optionally s and b) fields
mode        scalar                 1 = split each hex cell into 28 tets (default)
```

```
                                    2 = split each hex cell into 6 tets

                                                                                .
Output      Value/(Size)            Description
--------------------------------------------------------------------------------
sout        struct                  Output grid or problem struct
```

See also

quad2tri, tet2hex, tri2quad

11.269 grid/holegrid

HOLEGRID Generate 2d rectangular grid with a circular hole.

[GRID] =

HOLEGRID(NS, NR, XP, R, XC, TH_OFFSET) Generates a grid for a rectangular domain with a circular hole. NS specifies the cell resolution of the outer square (default 5), and NR the number of cells in the radial direction of the outer layer (default 5). XP=
[x1 y1 x2 y2] specifies the coordinates of the outer lower left (x1,y1) and upper right (x2,y2) corners.The optional arguments XC=
[x_c0;y_c0] and R specify the center coordinates and radius of the circle (default R=0.5 and XC=
[0;0]). Furthermore, TH_OFFSET specify a rotation of the whole grid.

Examples

```
1) A 2x2 square with a circular hole with radius 0.5.

grid = holegrid( 5, 5, [-1 1;-1 1], 0.5, [0;0] );

2) Rotated grid with offset hole.

grid = holegrid( 5, 5, [-1 1;-1 1], 0.5, [0.25;0], pi/4 );
```

See also

blockgrid, circgrid, cylgrid, linegrid, rectgrid, ringgrid, spheregrid

11.270 grid/linegrid

LINEGRID Generate 1D line grid.

[GRID] =

LINEGRID(P, XMIN, XMAX) Generates a regular 1D line grid with coordinates in P. If P is an integer scalar {P=10 per default} P equally spaced cells will be generated on a line spanning XMIN and XMAX. If the optional arguments (only used if P is a scalar) XMIN and XMAX are omitted they default to 0 and 1 respectively.

Examples

1) 10 grid cells on a line spanned by[0..1]

```
grid = linegrid();
```

2) 25 grid cells on a line spanned by[-0.5..0.5]

```
grid = linegrid( 25, -0.5, 0.5 );
```

3) 10 grid cells on a log spaced line[-2..2]

```
p = logspace(-2,2,11);
grid = linegrid( p );
```

See also

blockgrid, circgrid, cylgrid, holegrid, rectgrid, ringgrid, spheregrid

11.271 grid/mesh_boundaries

MESH_BOUNDARIES Mesh boundaries.

[P_E, E] =

MESH_BOUNDARIES(B, HMAX_E) Mesh boundaries

11.272 grid/mesh_size

MESH_SIZE Define mesh sizes.

[HMAX, HMAXB, HMAXP] =

MESH_SIZE(HMAX, HMAXB, N_SDIM, E, F, V, DO_AVG) Define mesh sizes.

11.273 grid/rectgrid

RECTGRID Generate 2D rectangular tensor product quadrilateral grid.

[GRID] =

RECTGRID(N_CX, N_CY, XP) Generates a tensor product quadrilateral grid with N_CX, N_CY cells in the x and y-directions, respectively. The optional argument XP=
[x1 y1 x2 y2] specifies the coordinates of the lower left (x1,y1) and upper right (x2,y2) corners. A call without input arguments generates a default 10x10 unit square.

Examples

1) A 10x10 grid on the domain spanned by[0..1, 0..1]

```
grid = rectgrid();
```

2) A 20x40 grid on the domain spanned by[-0.5..0.5, 1..2]

```
grid = rectgrid( 20, 40, [-0.5 0.5; 1 2] );
```

See also

blockgrid, circgrid, cylgrid, holegrid, linegrid, ringgrid, spheregrid

11.274 grid/reorient_cells

REORIENT_CELLS Reorient cells to corrent orientation.

[GRID] =

REORIENT_CELLS(GRID, DO_ALL, DO_CLEANUP, I_CHECK) Reorient cells to correct orientation. DO_ALL toggles all permutations while otherwise it is assumed cells are not inverted or othewise have incorrect shapes. DO_CLEANUP indicates if grid cleanup and repair should be attempted. I_CHECK is a flag to choose check/ test type (1 = test Jacobian determinant (default), 2 = test if polygon points are in clockwise order, 3 = split hexahedra into six tetrahedra).

See also

gridcheck

11.275 grid/ringgrid

RINGGRID Generate 2d grid of a ring.

[GRID] =

RINGGRID(NR, NTH, RI, RO, XP, TH_START, TH_END) Generates a quadrilateral grid for a ring shaped domain with NR cells in the radial and NTH cells in the tangential directions, respectively. RI and RO give the inner and outer radius, while the optional arguments XP, TH_START, and TH_END specify the center coordinates start, and end angles. Furthermore, the position of the radial layers can be manually specified through a vector of coordinates in NR (which overrides RI and RO).

Examples

```
1) An annulus 64 cell grid for a circle with radius 1 and
```

outer radius 3 centered at[0.5, 0.5]

```
grid = ringgrid( 10, 36, 1, 3, [0.5;0.5] );

2) An annulus with prescribed radial layers.

grid = ringgrid( [0.05 0.06 0.08 0.11 0.15], 32, [], [], [0;0], -3/4*pi );

3) Quarter ring.

grid = ringgrid( [0.05 0.06 0.08 0.11 0.15], 32, [], [], [0;0], 0, pi/2 );
```

See also

blockgrid, circgrid, cylgrid, holegrid, linegrid, rectgrid, spheregrid

11.276 grid/project_bdrp

PROJECT_BDRP Project boundary points to geometric boundary.

[GRID] =

PROJECT_BDRP(GRID, GEOM, MERGE) Projects boundary nodes according to the boundaries specified in the GEOM struct. If the MERGE flag is set then all geometry objects will be merged.

11.277 grid/quad2tri

QUAD2TRI Convert quadrilaterals to triangles.

[SOUT] =

QUAD2TRI(SIN) Convert quadrilateral to triangular cells.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
sin         struct                 Grid or problem struct with
                                   p, c (and optionally s and b) fields
iorder      0/{1}                  Node ordering of triangles.
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
```

```
sout        struct                  Output grid or problem struct
```

See also

hex2tet, tet2hex, tri2quad

11.278 grid/quadmesh

QUADMESH Generate quadrilateral mesh using distance functions.

[P, C] =

QUADMESH(FD, H0, BBOX, PFIX, HTOL, NCOFF) Function to generate a grid with quadrilateral cells for geometry objects defined using distance functions. Accepts the following input and output arguments.

```
Property    Value/{Default}           Description
-------------------------------------------------------------------------------
fd          anonymous function        Distance function
h0          scalar  {0.1}             Target grid size
bbox        array   [2,2]             Bounding box enclosing geometry object
pfix        array   [n_pfix,2]        Points that must be in the grid (optional)
htol        scalar  {0.2}             Cell size width fraction in grid snapping
ncoff       integer {2}               Number of extra cells to offset in
                                      surrounding rectangular grid construction
p_intf      array  [2*n_l+nan,2]      Interface line segemet array (line format)
p           array  [n_p,2]            Output array of grid vertices
c           array  [n_c,4]            Output array of quadrilateral connectivities
```

Example: (Uniform mesh on unit circle) fd = @(p) sqrt(sum(p.^2,2)) - 1;
[p,c] =

quadmesh(fd, 0.1,
[-1 -1;1 1]); patch(`vertices`,p,`faces`,c,`facecolor`,g), axis equal

Example: (Ellipse) fd = @(p) p(:,1).^2/2^2 + p(:,2).^2/1^2 - 1;
[p,c] =

quadmesh(fd, 0.1,
[-2 -1;2 1]); patch(`vertices`,p,`faces`,c,`facecolor`,g), axis equal

Example: (Rectangle with circular hole) ddiff = @(d1,d2) max(d1,-d2); drectangle = @(p,x1,x2,y1,y2) -min(min(min(-y1+p(:,2),y2-p(:,2)),-x1+p(:,1)),x2-p(:,1)); dcircle = @(p,xc,yc,r) sqrt((p(:,1)-xc).^2+(p(:,2)-yc).^2)-r; fd = @(p) ddiff(drectangle(p,-1,1,-1,1),dcircle(p,0,0,0.5));

[p,c] =
quadmesh(fd, 0.025,
[-1,-1;1,1],
[-1,-1;-1,1;1,-1;1,1]); patch(vertices,p,faces,c,facecolor,g), axis equal

See also

gridgen_quad, gridalign2

11.279 grid/reconstruct_bdr

ASSIGN_BOUNDARIES Assign and reconstruct boundary struct.

[B] =

RECONSTRUCT_BDR(GRID, IND_B, EFV_B) Reconstructs boundary struct from the GRID struct and bounday numbers in IND_B which correspond to the edge/face vertex ordering in EFV_B.

11.280 grid/selcells

SELCELLS Select cells from expression.

[CIND] =

SELCELLS(SIN, S_EXPR) Returns index to cells in selection expression. If a sdim field is not found in the imput struct the default coordinate names x, y, and z will be used.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
sin         struct                 Grid or problem struct with
                                   p, c (and s) fields
s_expr      string                 Cell selection string expression
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
cind        array                  Array with indicies to cells which cell
                                   vertices satisfy the selection expression
```

Example

Given a problem struct fea, select cells between coordinates 0.2<x<0.5

```
cind = selcells( fea, '(x>0.2).*(x<0.5)' );
```

11.281 grid/spheregrid

SPHEREGRID Generate 3d hexahedral grid for a sphere.

[GRID] =

SPHEREGRID(NS, NR, R, XP) Generates a hexahedral grid for a spherical domain with NS+NR cells in the radial direction. NS specifies the cell resolution of the inner cube (default 4), and NR the number of cells in the radial direction of the outer layer (default 3). The optional arguments R and XP =
[x0;y0;z0] specify the radius and center coordinates of the sphere (default R = 1 and XP = [0;0;0]).

Examples

1) Generate a grid for a sphere with radius 1 centered at[0, 0]

```
grid = spheregrid();
```

2) Generate a grid for a sphere with radius 0.5 centered at[1, 1, 1]

```
grid = spheregrid( 16, 12, 0.5, [1;1;1] );
```

See also

blockgrid, circgrid, cylgrid, holegrid, linegrid, rectgrid, ringgrid

11.282 grid/searchgraph

SEARCHGRAPH Find path from two points in edge graph.

[G] =

SEARCHGRAPH(I1, I2, E, ITMAX, N_MAX_BRANCH) Traverse graph E from point I1 to I2 and returns the path found in G. ITMAX (default 100) sets the maximum number of iterations and N_MAX_BRANCH the maximum number of branches to search.

11.283 grid/tet2hex

TET2HEX Convert tetrahedra to hexahedra.

[SOUT] =

TET2HEX(SIN) Convert tetrahedral to hexahedral cells.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
sin         struct                 Grid or problem struct with
                                   p, c (and optionally s and b) fields
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
sout        struct                 Output grid or problem struct
```

See also

hex2tet, quad2tri, tri2quad

11.284 grid/tri2quad

TRI2QUAD Convert triangles to quadrilaterals.

[SOUT] =

TRI2QUAD(SIN) Convert triangular to quadrilateral cells.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
sin         struct                 Grid or problem struct with
                                   p, c (and optionally s and b) fields
```

```
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
sout        struct                 Output grid or problem struct
```

See also

hex2tet, quad2tri, tet2hex

11.285 gridgen

GRIDGEN Grid generation for geometry objects.

[GRID, STATS] =

GRIDGEN(SIN, VARARGIN) Function to generate a grid for geometry objects by calling an external grid generation algorithm. SIN is a valid fea problem struct or cell array or geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}          Description
-------------------------------------------------------------------------------
```

meshgen string {distmesh} Grid generation algorithm

gmsh, distmesh, or triangle hmax scalar/arr {0.1} Target grid size for subdomains hmaxb scalar/arr {
[]} Target grid size for boundaries avhb logical {true} Average hmax to boundaries fixpnt array
[n_p,n_sdim] Array of (fixed) points that must be present nsm scalar {3} Number of (post) grid smoothing steps intb logical {false} Keep/reconstruct internal boundaries waitbar scalar {0} Show/hide waitbar fid scalar {1} File identifier for output (
[]=no output)

MESHGEN specifies which mesh generation algorithm to use (default distmesh) and calls the corresponding grid generation code.

HMAX indicates target grid cell diameters, and is either a numeric scalar prescribing the grid size for the entire geometry, or an array with HMAX values corresponding to individual subdo-

mains. Positive HMAX values uses the minimum mesh size for shared boundaries, while negative applys the mean value.

HMAXB is analogous to HMAX but related to boundaries (edges/faces). HMAXB can be a single scalar applicable to all boundaries, a numeric array with entries corresponding to individual boundaries.

The AVHB logical flag toggles if internal boundaries inheriting HMAX values (when HMAXB is unspecified) should be assigned the smallet HMAX value from neighbouring subdomains, or the mean value (default).

NSM (default 3) the number of GRIDSMOOTH smoothing steps to perform.

INTB toggels internal boundaries (in grid.b) on or off (default).

Examples

```
1) Unit square with uniform global grid size set to 0.1.

grid = gridgen( {gobj_rectangle()}, <tt>hmax</tt>, 0.1 );
plotgrid( grid )

2) Unit square with a finer grid along the top boundary.

grid = gridgen( {gobj_rectangle()}, <tt>hmax</tt>, 0.5, ...
                     <tt>hmaxb</tt>, [0.5 0.5 0.01 0.5] );
plotgrid( grid )

3) Domain with curved boundaries meshed with quadrilaterals.

geom.objects = {gobj_rectangle() gobj_circle([0 0],.6) gobj_circle([1 1],.3,<tt>C2</tt>)};
geom = geom_apply_formula( geom, <tt>R1-C1-C2</tt> );
grid = gridgen( geom, <tt>hmax</tt>, 0.1, <tt>quad</tt>, true );
plotgrid( grid )

4) Two connected subdomains with a shared boundary.

geom.objects = { gobj_polygon([-2e-3 -8e-3;0 -8e-3;0 -6e-3;0 6e-3;0 8e-3;-2e-3 8e-3]), ...
                 gobj_polygon([0 -6e-3;2e-3 -5e-3;2e-3 4e-3;0 6e-3]) };
hmax  = 5e-4;
hmaxb = hmax*ones(1,4);
hmaxb(3) = hmax/5;
grid  = gridgen( geom, <tt>hmax</tt>, hmax, <tt>hmaxb</tt>, hmaxb );
plotgrid( grid )

5) Composite component with several subdomains.

r1 = gobj_rectangle( 0, 0.11, 0, 0.12,  <tt>R1</tt> );
c1 = gobj_circle( [ 0.065 0 ],   0.015, <tt>C1</tt> );
```

```
c2 = gobj_circle( [ 0.11 0.12 ], 0.035, <tt>C2</tt> );
c3 = gobj_circle( [ 0 0.06 ],    0.025, <tt>C3</tt> );
r2 = gobj_rectangle( 0.065, 0.16, 0.05, 0.07, <tt>R2</tt> );
c4 = gobj_circle( [ 0.065 0.06 ], 0.01, <tt>C4</tt> );
geom.objects = { r1 c1 c2 c3 r2 c4 };
geom = geom_apply_formula( geom, <tt>R1-C1-C2-C3</tt> );
geom = geom_apply_formula( geom, <tt>R2+C4</tt> );

grid  = gridgen( geom, <tt>hmax</tt>, [0.0025 0.05 0.0025] );
plotgrid( grid )

6) Complex geometry with several holes and subdomains.

w = 10e-4; L = 3*w; H = 5*w;
p1  = gobj_polygon( [w/10 0;(L-w/4)/2 0;(L-w/4)/2 H;0 H;0 H/3], <tt>P1</tt> );
p2  = gobj_polygon( [(L+w/4)/2 0;L 0;L H-H/3;L H;(L+w/4)/2 H], <tt>P2</tt> );
r1  = gobj_rectangle( (L-w/4)/2, (L+w/4)/2, 0, H, <tt>R1</tt> );
c1  = gobj_circle( [2*w/3 3*w], w/3, <tt>C1</tt> );
c2  = gobj_circle( [2*w/3 2*w], w/3, <tt>C2</tt> );
c3  = gobj_circle( [2*w/3 1*w], w/3, <tt>C3</tt> );
c4  = gobj_circle( [L-w/2 4.5*w], w/8, <tt>C4</tt> );
c5  = gobj_circle( [L-w   4.5*w], w/8, <tt>C5</tt> );
c6  = gobj_circle( [L-w/2 4*w], w/8, <tt>C6</tt> );
c7  = gobj_circle( [L-w   4*w], w/8, <tt>C7</tt> );
c8  = gobj_circle( [L-w/2 3.5*w], w/8, <tt>C8</tt> );
c9  = gobj_circle( [L-w   3.5*w], w/8, <tt>C9</tt> );
c10 = gobj_circle( [L-w/2 3*w], w/8, <tt>C10</tt> );
c11 = gobj_circle( [L-w   3*w], w/8, <tt>C11</tt> );

geom.objects = { p1 p2 r1 c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 };
geom = geom_apply_formula( geom, <tt>P1-C1-C2-C3</tt> );
geom = geom_apply_formula( geom, <tt>P2-C4-C5-C6-C7-C8-C9-C10-C11</tt> );

hmaxb = zeros(1,21);
hmaxb([5 20]) = w/50;
grid = gridgen( geom, <tt>hmax</tt>, w./[5 5 20], <tt>hmaxb</tt>, hmaxb );
plotgrid( grid )
```

See also

gridgen_distmesh, gridgen_gmsh, gridgen_triangle, gridgen_quad

11.286 grid/gridgen_distmesh

GRIDGEN_DISTMESH Generate unstructured grid for geometry objects.

[GRID, STATS] =

GRIDGEN_DISTMESH(SIN, VARARGIN) Function to generate a grid for geometry objects by calling the exernal DISTMESH library. SIN is a valid fea problem struct or cell array or geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}           Description
-------------------------------------------------------------------------------
hmax        scalar/arr {0.1}          Target grid size for subdomains
hmaxb       scalar/arr {[]}           Target grid size for boundaries
avhb        logical {true}            Average hmax to boundaries
itmax       scalar {1000}             Maximium number of iterations
fixpnt      array  [n_p,n_sdim]       Array of (fixed) points that must be present
nsmh        scalar {50}               Mesh size function smoothing steps
hdfcn       function handle {[]}      Custom mesh size distribution function
waitbar     scalar {false}            Show waitbar
fid         scalar {1}                File identifier for output ([]=no output)
```

HMAX indicates target grid cell diameters, and is either a numeric scalar prescribing the grid size for the entire geometry, or an array with HMAX values corresponding to individual subdomains.

HMAXB is analogous to HMAX but related to boundaries (edges). HMAXB can be a single scalar applicable to all boundaries, a numeric array with entries corresponding to individual boundaries, or a cell array. In the case HMAXB is a cell array each entry and boundary can be prescribed a value for the mean cell edge size or a numeric array of grid point distributions (spanning 0 to 1) along the boundary.

AVHB determines if hmax from adjacent subdomains should be averaged to non-specified HMAXB entries (alternatively the minimum HMAX value is assigned).

NSMH (default 50) the number smoothing steps used in computation of non-uniform mesh size function.

See also

gridgen, distmesh

11.287 grid/gridgen_gmsh

GRIDGEN_GMSH Generate grid with Gmsh.

[GRID, STATS] =

GRIDGEN_GMSH(SIN, VARARGIN) Function to generate an unstructured GRID for FEATool

geometry objects by calling the external Gmsh mesh generator (http://gmsh.info and http://gmsh.info/doc/texinfo/gmsh.html)

If the Gmsh binary is not present and found in the INSTDIR the function will attempt to automatically download, and install it in the given directory. If this fails, please manually download and install Gmsh from the source reference and link above.

SIN is a valid FEATool fea problem, geometry, or geo struct, or alternatively a cell array of geometry objects. Accepts the following property/value pairs.

Property	Value/{Default}	Description
hmax	scal/arr {0.1}	Target grid size for geometry/subdomains
hmaxb	scal/arr {[]}	Target grid size for boundaries
hmaxp	scal/arr {[]}	Target grid size for vertices
nsm	scalar {3}	Number of grid smoothing steps
nref	scalar {0}	Number of uniform grid refinements
algo2	scalar {2}	2D mesh algorithm (1=MeshAdapt, 2=Automatic, 5=Delaunay, 6=Frontal, 7=BAMG, 8=DelQuad)
algo3	scalar {1}	3D mesh algorithm (1=Del, 2=New Del, 4=Front 5=Front Del, 6=Front Hex, 7=MMG3D, 9=R-tree)
quad	boolean {false}	Use quad meshing (for 2D)
tol	scalar {eps*1e3}	Deduplication tolerance
compound	boolean {true}	Use Gmsh compound boundaries
mshopt	cell {}	Cell array of Gmsh options
syscmd	string {<tt>default</tt>}	Gmsh system call command (default <tt>gmsh fdir/fname.geo -</tt>)
fname	string {<tt>fea_gmsh_UID</tt>}	Gmsh imp/exp file name (root)
fdir	string {tempdir}	Directory to write help files
clean	boolean {true}	Delete (clean) Gmsh help files
instdir	string {<tt>lib/gmsh</tt>}	Gmsh binary installation directory
fid	scalar {1}	File identifier for output ([]=no output)

HMAX indicates target grid cell diameters, and is either a numeric scalar prescribing the grid size for the entire geometry, or an array with HMAX values corresponding to individual subdomains. Positive HMAX values uses the minimum mesh size for shared boundaries, while negative applys the mean value.

HMAXB is analogous to HMAX but related to boundaries (edges/faces). HMAXB can be a single scalar applicable to all boundaries, a numeric array with entries corresponding to individual boundaries.

NSM (default 3) the number of post smoothing steps to perform.

NREF (default 0) the number of post uniform grid refinement steps.

ALGO2 and ALGO3 the Gmsh 2D and 3D mesh generation algorithms.

QUAD (default 0) toggles Blossom-Quad conversion for 2D geometries.
Additional Gmsh options can be provided with the cell array MSHOPT.

For example MSHOPT could be given as

{{Mesh, CharacteristicLengthMax, 1}, {Mesh, AnisoMax, 10}}

Examples

```
1) Unit square with uniform global grid size set to 0.1.

grid = gridgen_gmsh( {gobj_rectangle()}, <tt>hmax</tt>, 0.1 );
plotgrid( grid )

2) Unit square with a finer grid along the top boundary.

grid = gridgen_gmsh( {gobj_rectangle()}, <tt>hmax</tt>, 0.5, ...
                     <tt>hmaxb</tt>, [0.5 0.5 0.01 0.5] );
plotgrid( grid )

3) Domain with curved boundaries meshed with quadrilaterals.

geom.objects = {gobj_rectangle() gobj_circle([0 0],.6) gobj_circle([1 1],.3,<tt>C2</tt>)};
geom = geom_apply_formula( geom, <tt>R1-C1-C2</tt> );
grid = gridgen_gmsh( geom, <tt>hmax</tt>, 0.1, <tt>quad</tt>, true );
plotgrid( grid )

4) Two connected subdomains with a shared boundary.

geom.objects = { gobj_polygon([-2e-3 -8e-3;0 -8e-3;0 -6e-3;0 6e-3;0 8e-3;-2e-3 8e-3]), ...
                 gobj_polygon([0 -6e-3;2e-3 -5e-3;2e-3 4e-3;0 6e-3]) };
hmax  = 5e-4;
hmaxb = hmax*ones(1,4);
hmaxb(3) = hmax/5;
grid  = gridgen_gmsh( geom, <tt>hmax</tt>, hmax, <tt>hmaxb</tt>, hmaxb );
plotgrid( grid )

5) Composite component with several subdomains.

r1 = gobj_rectangle( 0, 0.11, 0, 0.12,  <tt>R1</tt> );
c1 = gobj_circle( [ 0.065 0 ],   0.015, <tt>C1</tt> );
c2 = gobj_circle( [ 0.11 0.12 ], 0.035, <tt>C2</tt> );
c3 = gobj_circle( [ 0 0.06 ],    0.025, <tt>C3</tt> );
r2 = gobj_rectangle( 0.065, 0.16, 0.05, 0.07, <tt>R2</tt> );
c4 = gobj_circle( [ 0.065 0.06 ], 0.01, <tt>C4</tt> );
geom.objects = { r1 c1 c2 c3 r2 c4 };
geom = geom_apply_formula( geom, <tt>R1-C1-C2-C3</tt> );
geom = geom_apply_formula( geom, <tt>R2+C4</tt> );
```

```
grid  = gridgen_gmsh( geom, <tt>hmax</tt>, [0.0025 0.05 0.0025] );
plotgrid( grid )

6) Complex geometry with several holes and subdomains.

w = 10e-4; L = 3*w; H = 5*w;
p1  = gobj_polygon( [w/10 0;(L-w/4)/2 0;(L-w/4)/2 H;0 H;0 H/3], <tt>P1</tt> );
p2  = gobj_polygon( [(L+w/4)/2 0;L 0;L H-H/3;L H;(L+w/4)/2 H], <tt>P2</tt> );
r1  = gobj_rectangle( (L-w/4)/2, (L+w/4)/2, 0, H, <tt>R1</tt> );
c1  = gobj_circle( [2*w/3 3*w], w/3, <tt>C1</tt> );
c2  = gobj_circle( [2*w/3 2*w], w/3, <tt>C2</tt> );
c3  = gobj_circle( [2*w/3 1*w], w/3, <tt>C3</tt> );
c4  = gobj_circle( [L-w/2 4.5*w], w/8, <tt>C4</tt> );
c5  = gobj_circle( [L-w   4.5*w], w/8, <tt>C5</tt> );
c6  = gobj_circle( [L-w/2 4*w], w/8, <tt>C6</tt> );
c7  = gobj_circle( [L-w   4*w], w/8, <tt>C7</tt> );
c8  = gobj_circle( [L-w/2 3.5*w], w/8, <tt>C8</tt> );
c9  = gobj_circle( [L-w   3.5*w], w/8, <tt>C9</tt> );
c10 = gobj_circle( [L-w/2 3*w], w/8, <tt>C10</tt> );
c11 = gobj_circle( [L-w   3*w], w/8, <tt>C11</tt> );

geom.objects = { p1 p2 r1 c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 };
geom = geom_apply_formula( geom, <tt>P1-C1-C2-C3</tt> );
geom = geom_apply_formula( geom, <tt>P2-C4-C5-C6-C7-C8-C9-C10-C11</tt> );

hmaxb = zeros(1,21);
hmaxb([5 20]) = w/50;
grid = gridgen_gmsh( geom, <tt>hmax</tt>, w./[5 5 20], <tt>hmaxb</tt>, hmaxb );
plotgrid( grid )
```

See also

gridgen, impexp_gmsh, geom2geo

11.288 grid/gridgen_quad

GRIDGEN_QUAD Generate quadrilateral grid for a geometry object.

[GRID] =

GRIDGEN_QUAD(SIN, VARARGIN) Function to generate a grid with quadrilateral cells for geometry objects. SIN is a valid fea problem struct or cell array or geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}          Description
-------------------------------------------------------------------------------
hmax        scalar  {0.1}            Target grid size
htol        scalar  {0.2}            Cell size width fraction in grid snapping
ncoff       integer {2}              Number of extra cells to offset in
                                     surrounding rectangular grid construction
nsm         integer {20}             Number of grid smoothing steps
igobj       integer {1}              Geometry object for which to create grid
fid         scalar  {1}              File identifier for output ([]=no output)
```

Example: (Circle) fea.geom.objects = { gobj_circle() }; fea.grid = gridgen_quad(fea, hmax, 0.1); plotgrid(fea)

Example: (Square with circular hole) fea.geom.objects = { gobj_rectangle() gobj_circle([0.5 0.5],0.25) }; fea = geom_apply_formula(fea, R1 - C1); fea.grid = gridgen_quad(fea, hmax, 0.025); plotgrid(fea)

Example: (Star shaped polygon) pv = [-0.4 -0.5;0.4 -0.2;0.4 -0.7;1.5 -0.4; 0.9 0.1 ; 1.6 0.8;0.5 0.5;0.2 1.0;0.1 0.4;-0.7 0.7]; fea.geom.objects = { gobj_polygon(pv) }; fea.grid = gridgen_quad(fea, hmax, 0.05); plotgrid(fea)

Example: (Ellipse) fea.geom.objects = { gobj_ellipse([0 0], 2, 1) }; fea.grid = gridgen_quad(fea, hmax, 0.1); plotgrid(fea)

Example: (Polygon with ellipse) g1 = gobj_polygon([-0.77 1.1 1 0 -1; -1 -1.2 1 2 1]'); g2 = gobj_ellipse([0 0], 2, 0.5); fea.geom.objects = { g1 g2 }; fea = geom_apply_formula(fea, P1 + E1); fea.grid = gridgen_quad(fea, hmax, 0.05); plotgrid(fea)

Example: (Polygon with rounded corner and rectangular hole) g1 = gobj_polygon([0.2 1 1 0 -0.1 0 0.3; 0 0 1 1 1 0.2 0.3]'); g2 = gobj_circle([1 1], 0.4); g3 = gobj_rectangle(0.1, 0.4, 0.6, 0.8); fea.geom.objects = { g1 g2 g3 }; fea = geom_apply_formula(fea, P1 - C1 - R1); fea.grid = gridgen_quad(fea, hmax, 0.05); plotgrid(fea)

See also

quadmesh, gridalign2

11.289 grid/gridgen_triangle

GRIDGEN_TRIANGLE Generate unstructured 2D grid with Triangle.

[GRID, STATS] =

GRIDGEN_TRIANGLE(SIN, VARARGIN) Function to generate an unstructured 2D triangular GRID for FEATool geometry objects by calling the exernal Triangle grid generator library (https://www.cs.cmu.edu/~quake/triangle.html)

```
J R Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and
Delaunay Triangulator, in Applied Computational Geometry: Towards
Geometric Engineering (Ming C. Lin and Dinesh Manocha, editors),
volume 1148 of Lecture Notes in Computer Science, pages 203-222,
Springer-Verlag, Berlin, May 1996. (From the First ACM Workshop on
Applied Computational Geometry.)
```

If the Triangle binary is not present and found in the INSTDIR the function will attempt to automatically download, (compile if required), and install it in the given directory. If this fails, please manually download and install Triangle from the source reference and link above.

SIN is a valid FEATool fea problem struct or cell array of geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}            Description
-------------------------------------------------------------------------------
hmax        scal/arr {0.1}             Target grid size for geometry/subdomains
hmaxb       scal/arr {[]}              Target grid size for boundary edges
q           scalar   {28}              Minimum target angle (quality)
syscmd      string   {<tt>default</tt>}       Triangle system call command (default
                                       <tt>triangle -I -q%f -j -e -a -A %s.poly</tt>)
compcmd     string   {<tt>default</tt>}       Triangle unix/linux compilation command
                                       (<tt>cc -O triangle.c -lm -o triangle</tt>)
fname       string   {<tt>fea_tri_UID</tt>}   Triangle imp/exp file name (root)
fdir        string   {tempdir}         Directory to write help files
clean       boolean  {true}            Delete (clean) Triangle help files
instdir     string   {<tt>lib/triangle</tt>}  Triangle installation directory
fid         scalar   {1}               File identifier for output ([]=no output)
```

HMAX indicates target grid cell diameters, and is either a numeric scalar prescribing the grid size for the entire geometry, or an array with HMAX values corresponding to individual subdomains.

HMAXB is analogous to HMAX but related to boundaries (edges). HMAXB can be a single scalar applicable to all boundaries, a numeric array with entries corresponding to individual boundaries, or a cell array. In the case HMAXB is a cell array each entry and boundary can be prescribed a value for the mean cell edge size or a numeric array of grid point distributions (spanning 0 to 1) along the boundary.

Q (default 28 degrees) specifies a minimum target angle (values less than 33 are generally acceptable, while higher values might prevent Triangle convergence).

Examples

```
1) Unit square with uniform global grid size set to 0.1.

grid = gridgen_triangle( {gobj_rectangle()}, <tt>hmax</tt>, 0.1 );
plotgrid( grid )

2) Unit square with prescribed edge node distribution.

flog  = @(n,m) ((logspace(0,1,n)-1)/9).^m;   % Log distribution function.
grid = gridgen_triangle( {gobj_rectangle()}, <tt>hmax</tt>, 0.5, ...
               <tt>hmaxb</tt>, {1-flog(50,2) flog(50,2) 1-flog(15,1.5) flog(15,1.5)} );
plotgrid( grid )

3) Domain with a fine grid along curved boundaries (number 5 and 6).

geom.objects = {gobj_rectangle() gobj_circle([0 0],.6) gobj_circle([1 1],.3,<tt>C2</tt>)};
geom = geom_apply_formula( geom, <tt>R1-C1-C2</tt> );
grid = gridgen_triangle( geom, <tt>hmax</tt>, .1, <tt>hmaxb</tt>, [.1 .1 .1 .1 .01 .01] );
plotgrid( grid )

4) Two connected subdomains with refined grid along the shared boundary (9).

geom.objects = { gobj_rectangle(-2e-3,0,-8e-3,8e-3), ...
                 gobj_polygon([0 -6e-3;2e-3 -5e-3;2e-3 4e-3;0 6e-3]) };
hmax  = 5e-4;
hmaxb = hmax*ones(1,9);
hmaxb(9) = hmax/5;
grid  = gridgen_triangle( geom, <tt>hmax</tt>, hmax, <tt>hmaxb</tt>, hmaxb );
plotgrid( grid )

5) Component with fine grid on subdomains 2 and 3, and curved boundaries.

r1 = gobj_rectangle( 0, 0.11, 0, 0.12,  <tt>R1</tt> );
c1 = gobj_circle( [ 0.065 0 ],   0.015, <tt>C1</tt> );
c2 = gobj_circle( [ 0.11 0.12 ], 0.035, <tt>C2</tt> );
c3 = gobj_circle( [ 0 0.06 ],    0.025, <tt>C3</tt> );
r2 = gobj_rectangle( 0.065, 0.16, 0.05, 0.07, <tt>R2</tt> );
c4 = gobj_circle( [ 0.065 0.06 ], 0.01, <tt>C4</tt> );
geom.objects = { r1 c1 c2 c3 r2 c4 };
geom = geom_apply_formula( geom, <tt>R1-C1-C2-C3</tt> );
geom = geom_apply_formula( geom, <tt>R2+C4</tt> );

hmax  = [0.02 0.0015 0.0015];
hmaxb = 0.02*ones(1,34);
hmaxb([8:12]) = 0.0025;
```

```
grid  = gridgen_triangle( geom, <tt>hmax</tt>, hmax, <tt>hmaxb</tt>, hmaxb );
plotgrid( grid )

6) Complex geometry with several holes and subdomains.

w = 10e-4; L = 3*w; H = 5*w;
p1  = gobj_polygon( [w/10 0;L 0;L H-H/3;L H;0 H;0 H/3], <tt>P1</tt> );
r1  = gobj_rectangle( (L-w/4)/2, (L+w/4)/2, -sqrt(eps), H, <tt>R1</tt> );
c1  = gobj_circle( [2*w/3 3*w], w/3, <tt>C1</tt> );
c2  = gobj_circle( [2*w/3 2*w], w/3, <tt>C2</tt> );
c3  = gobj_circle( [2*w/3 1*w], w/3, <tt>C3</tt> );
c4  = gobj_circle( [L-w/2 4.5*w], w/8, <tt>C4</tt> );
c5  = gobj_circle( [L-w   4.5*w], w/8, <tt>C5</tt> );
c6  = gobj_circle( [L-w/2 4*w], w/8, <tt>C6</tt> );
c7  = gobj_circle( [L-w   4*w], w/8, <tt>C7</tt> );
c8  = gobj_circle( [L-w/2 3.5*w], w/8, <tt>C8</tt> );
c9  = gobj_circle( [L-w   3.5*w], w/8, <tt>C9</tt> );
c10 = gobj_circle( [L-w/2 3*w], w/8, <tt>C10</tt> );
c11 = gobj_circle( [L-w   3*w], w/8, <tt>C11</tt> );
geom.objects = { p1 r1 c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 };
geom = geom_apply_formula( geom, <tt>P1-C1-C2-C3-C4-C5-C6-C7-C8-C9-C10-C11</tt> );

hmaxb = w/5*ones(1,59);
hmaxb([5 9]) = w/50;
grid  = gridgen_triangle( geom, <tt>hmax</tt>, w./[5 20 5], <tt>hmaxb</tt>, hmaxb );
plotgrid( grid )
```

See also

gridgen, impexp_triangle, geom2poly

11.290 grid/gridstat

GRIDSTAT Grid statistics.

[STAT, S] =

GRIDSTAT(GRID) Computes and returns grid statistics, such as the grid cell area and volume, and the quality measure; radius ratio for simplices and min/max diagonal ratio for quadrilaterals and hexahedrons.

11.291 install

INSTALL Add the FEATool toolbox to the Octave/Matlab path.

INSTALL adds the necessary folders to the Octave/Matlab path for the FEATool toolbox to be used from anywhere. Please read the included documentation by opening index.html in a web browser.

11.292 intsubd

INTSUBD Integate expression over subdomains.

[VAL] =

INTSUBD(S_EXPR, PROB, IND_S, IND_C, I_CUB, SOLNUM) Integrates the expression S_EXPR over the subdomains indicated in IND_S, or alternatively the cells in IND_C. PROB is a valid finite element problem struct, and I_CUB specifies the numerical integration rule.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
s_expr      string                 Expression to integrate
prob        struct                 Finite element problem struct
ind_s       [1,n_subd]             Subdomain numbers (default all)
ind_c       [1,n_cells]            Cell indices (default all)
i_cub       scalar                 Numerical integration rule (default 2)
solnum      scalar {n_sols}        Solution number/time to evaluate
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
val         scalar                 Result of integration
```

See also

intbdr, minmaxsubd, minmaxbdr

11.293 intbdr

INTBDR Integation of expression over boundaries.

[VAL] =

INTSUBD(S_EXPR, PROB, IND_B, I_CUB, SOLNUM) Integrates the expression S_EXPR over

the boundaries indicated in IND_B. PROB is a valid finite element problem struct, and I_CUB specifies the numerical integration rule.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
s_expr      string                 Expression to integrate
prob        struct                 Finite element problem struct
ind_b       [1,n_bdr]              Boundary numbers (default all)
i_cub       scalar                 Numerical integration rule (default 2)
solnum      scalar {n_sols}        Solution number/time to evaluate
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
val         scalar                 Result of integration
```

See also

intsubd, minmaxsubd, minmaxbdr

11.294 minmaxsubd

MINMAXSUBD Calculate subdomain minima and maxima.

[MIN_VAL, MAX_VAL, MIN_COORD, MAX_COORD] =

MINMAXSUBD(S_EXPR, PROB, IND_S, IND_C, I_CUB, SOLNUM) Evaluates the mimimum and maximum value of expression S_EXPR over the subdomains indicated in IND_S or alternatively the cells in IND_C. PROB is a valid finite element problem struct. Returns the minima and maxima in MIN_VAL and MAX_VAL, and the corresponding coordinates in MIN_COORD and MAX_COORD.

```
Input       Value/[Size]           Description
-------------------------------------------------------------------------------
s_expr      string                 Expression to evaluate
prob        struct                 Finite element problem struct
ind_s       [1,n_subd]             Subdomain numbers (default all)
ind_c       [1,n_cells]            Cell indices (default all)
i_cub       scalar                 Evaluation point rule (default 2)
solnum      scalar {n_sols}        Solution number/time to evaluate
                                                                              .
Output      Value/[Size]           Description
-------------------------------------------------------------------------------
min_val     scalar                 Minimum value of expression
max_val     scalar                 Maximum value of expression
```

See also

minmaxbdr, intsubd, intbdr

11.295 minmaxbdr

MINMAXBDR Calculate boundary minima and maxima.

[MIN_VAL, MAX_VAL, MIN_COORD, MAX_COORD] =

MINMAXBDR(S_EXPR, PROB, IND_B, I_CUB, SOLNUM) Evaluates the mimimum and maximum value of expression S_EXPR over the boundaries indicated in IND_B. PROB is a valid finite element problem struct. Returns the minima and maxima in MIN_VAL and MAX_VAL, and the corresponding coordinates in MIN_COORD and MAX_COORD.

```
Input       Value/[Size]           Description
-----------------------------------------------------------------------------
s_expr      string                 Expression to evaluate
prob        struct                 Finite element problem struct
ind_b       [1,n_bdr]              Boundary numbers (default all)
i_cub       scalar                 Evaluation point rule (default 2)
solnum      scalar {n_sols}        Solution number/time to evaluate
                                                                            .
Output      Value/[Size]           Description
-----------------------------------------------------------------------------
min_val     scalar                 Minimum value of expression
max_val     scalar                 Maximum value of expression
```

See also

minmaxsubd, intsubd, intbdr

11.296 physmodes/addphys

ADDPHYS Parse and add physics mode to problem struct.

[PROB] =

ADDPHYS(PROB, F_PHYS, C_DVAR, F_OUT) Parses and adds the physics mode with function

handle F_PHYS to the problem struct PROB. C_DVAR prescribes the given instead of default dependent variables names, and the F_OUT flag adds the mode to the PROB struct or optionally simply returns the computed physics mode.

See also

parsephys

11.297 physmodes/axistressstrain

AXISTRESSSTRAIN Axisymmetric stress-strain physics mode.

11.298 physmodes/brinkmaneqns

BRINKMANEQNS Brinkman equations physics mode.

11.299 physmodes/conductivemediadc

CONDUCTIVEMEDIADC Conductive media DC physics mode.

11.300 physmodes/eulerbeam

EULERBEAM 1D Euler-Bernoulli beam physics mode.

11.301 physmodes/electrostatics

ELECTROSTATICS Electrostatics physics mode.

11.302 physmodes/magnetostatics

MAGNETOSTATICS Magnetostatics physics mode.

11.303 physmodes/convectiondiffusion

CONVECTIONDIFFUSION Convection and diffusion physics mode definition.

11.304 physmodes/customeqn

CUSTOMEQN Custom equation physics mode definition.

11.305 physmodes/darcyslaw

DARCYSLAW Darcy's law physics mode.

11.306 physmodes/heattransfer

HEATTRANSFER Heat transfer physics mode definition.

11.307 physmodes/linearelasticity

LINEARELASTICITY Linear elasticity physics mode.

11.308 physmodes/list_physmodes

LIST_PHYSMODES Lists and orders the available physics modes.

11.309 physmodes/navierstokes

NAVIERSTOKES Navier-Stokes equations physics mode.

11.310 physmodes/physmode

PHYSMODE Finishes a physics mode and returns the phys struct.

11.311 physmodes/planestrain

PLANESTRAIN Plane strain physics mode.

11.312 physmodes/planestress

PLANESTRESS Plane stress physics mode.

11.313 physmodes/poisson

POISSON Poisson equation physics mode definition.

11.314 plotbdr

PLOTBDR Boundary plot function.

[H] =

PLOTBDR(PROB, VARARGIN) Plot and highlight boundaries. PROB is a valid finite element problem structure. Accepts the following property/value pairs.

```
Property    Value/{Default}          Description
-------------------------------------------------------------------------------
axis        off|{on}                 Show axes
grid        on |{off}                Show grid
normals     on |{off}                Show normals
axequal     off|{on}                 Axis equal setting
bbox        {0.05}                   Size of bounding box (0=off)
colors      {r,g,b,c,m,y}            Cell array with colors
labels      on/{off}                 Print boundary numbers
fontsize    {2*axes default}         Font size used in text labels
view        {[-35 20]}               3D view setting
alpha       {1}                      Transparency level
selbdr      {all}                    Index vector to boundaries to plot
parent      {gca}                    Axes handle to plot in
```

See also

plotgeom, plotgrid, plotsubd, postplot

11.315 plotgeom

PLOTGEOM Visualization of geometry objects.

[H] =

PLOTGEOM(SIN, VARARGIN) Function to plot and visualize geometry objects. SIN is a valid fea problem struct or cell array of geometry objects. Accepts the following property/value pairs.

```
Property    Value/{Default}        1D 2D 3D  Description
-------------------------------------------------------------------------------
facecolor   [.8 .8 1]              x  x  x   Face style or fixed color
edgecolor   {k}                    x  x  x   Edge line color
linestyle   {-}                    x  x  x   Edge line style
linewidth   {1}                    x  x  x   Edge line width
highlight   {}                     x  x  x   Highlight selected geometry objects
labels      off|{on}               x  x  x   Show geometry labels
fontsize    {axes default}         x  x  x   Font size used in labels
axequal     off|{on}               x  x  x   Axis equal setting
bbox        {0.05}                 x  x  x   Size of bounding box (0=off)
ngrid       {25}                   x  x  x   Background grid resolution
style       {1}                       x      Plot objects (=1) or boundaries (=2)
alpha       {0.5}                        x   Transparency level
parent      {gca}                  x  x  x   Plot axes handle
```

See also

plotgrid, plotbdr, plotsubd, postplot

11.316 plotgrid

PLOTGRID Grid plot function.

[H] =

PLOTGRID(PROB, VARARGIN) Function to plot and visualize a grid. PROB is a valid grid or problem struct. Accepts the following property/value pairs.

```
Property    Value/{Default}          Description
```

```
-------------------------------------------------------------------------------------
axis         off|{on}                  Show axes
grid         on |{off}                 Show grid
axequal      off|{on}                  Axis equal setting
edges        on |{off}                 Show internal edges (3D only)
view         {[45 45]}                 3D view setting
bbox         {0.05}                    Size of bounding box (0=off)
alpha        {1}                       Transparency level
facecolor    {[.9 .9 .9]}              Cell face style or fixed color
linestyle    {-}                       Line style
linewidth    {1}                       Line width
edgecolor    {b}                       Line color
edgecolor    {[.7 .7 .7]}              Cell line color
marker       {.}                       Node marker style
markercolor  {b}                       Node marker color
cellabels    on|{off}                  Show cell numbers
nodelabels   on|{off}                  Show vertex/node numbers
fontsize     {axes default}            Font size used in text
selcells     {all}                     Index vector to cells to plot
selsubd      {all}                     Index vector to subdomains to plot
renderer     {empty string}|plotly     Render in Matlab/Octave or Plotly
parent       {gca}                     Plot axes handle
setaxes      {on}|off                  Use axes modification settings
```

See also

plotbdr, plotsubd, postplot

11.317 plotsubd

PLOTSUBD Subdomain plotting function.

[H] =

PLOTSUBD(PROB, VARARGIN) Function to plot and highlight subdomains. PROB is a valid finite element problem struct. Accepts the following property/value pairs.

```
Property     Value/{Default}           Description
-------------------------------------------------------------------------------------
selsubd      {all}                     Index vector to subdomains to plot
axis         off|{on}                  Show axes
grid         on |{off}                 Show grid
normals      on |{off}                 Show normals
axequal      off|{on}                  Axis equal setting
bbox         {0.05}                    Size of bounding box (0=off)
labels       on/{off}                  Print boundary numbers
```

```
fontsize    {2*axes default}          Font size used in text labels
view        {[-35 20]}               3D view setting
alpha       {0.3}                    Transparency level
parent      {gca}                    Axes handle to plot in
```

See also

plotgrid, plotbdr, postplot

11.318 post/contour2

CONTOUR2 Generate contours on unstructured triangles.

[X, Y, GROUPS, IX_T] =

CONTOUR2(P, T, CDATA, LEV, DO_GROUPS) Generates contour line data for isovalue LEV on triangular cells defined by P and T, with CDATA value in the nodes P. DO_GROUPS indicates if groupings of connected line segments should be computed and returned as a cell array GROUPS of index arrays to X and Y. IX_T are indices pointing out which grid cell in T each countour segment in X/Y originates from.

11.319 post/postplot1

POSTPLOT1 1D Postprocessing function.

[H] =

POSTPLOT1(PROB, VARARGIN) 1D Postprocessing function. PROB is a valid problem struct. Outputs an array H of plot object handles. Accepts the following property/value pairs.

```
Property    Value/{Default}           Description
-------------------------------------------------------------------------------
surfexpr    string expression         Surface expression to plot
solnum      scalar {n_sols}           Solution number/time to plot
evalstyle   exact|{average}           Shape function evaluation type
color       {k}                       Line color
linestyle   {-}                       Line style
linewidth   {1}                       Line width
marker      {none}                    Marker style
markersize  {5}                       Marker size
markercolor {k}                       Marker color
title       string                    Plotly plot title
renderer    {empty string}|plotly     Render in Matlab/Octave or Plotly
```

```
cinds       {}                      Cell/subdomain plot indices
parent      {gca}                   Plot axes handle
```

11.320 post/postplot2

POSTPLOT2 2D Postprocessing function.

[H] =

POSTPLOT2(PROB, VARARGIN) 2D Postprocessing function. PROB is a valid problem struct. Outputs an array H of plot object handles. Accepts the following property/value pairs.

```
Property    Value/{Default}           Description
-------------------------------------------------------------------------------
surfexpr    string expression         Surface expression to plot
surfhexpr   string expression         Expression for surface height plot
isoexpr     string expression         Contour/iso-surface expression to plot
isolev      scalar {10}|vector        Number or vector of contour/iso-levels
isostyle    {<tt>-</tt>}                    Contour/iso-line style
isocolor    {<tt>k</tt>}                    Contour/iso-line color
arrowexpr   2 string expressions      Arrow expressions to plot
arrowspacing vector {[10 10]}         Number of arrows in each direction
arrowcolor  {r}                       Arrow color
solnum      scalar {n_sols}           Solution number/time to plot
evalstyle   exact|{average}           Shape function evaluation type
reflev      scalar {1}                Grid refinement level for exact evalstyle
facecolor   {interp}                  Cell face style or fixed color
linestyle   {none}                    Cell edge line style
linewidth   {1}                       Line width
edgecolor   {k}                       Cell edge line color
boundary    off|{on}                  Show geometry boundary
title       string                    Plotly plot title
renderer    {empty string}|plotly     Render in Matlab/Octave or Plotly
parent      {gca}                     Axes handle to plot in
```

11.321 post/postplot3

POSTPLOT3 3D Postprocessing function.

[H] =

POSTPLOT3(PROB, VARARGIN) 3D Postprocessing function. PROB is a valid problem struct. Outputs an array H of plot object handles. Accepts the following property/value pairs.

```
Property    Value/{Default}           Description
-------------------------------------------------------------------------------
surfexpr    string expression         Surface expression to plot
edges       on |{off}                 Show internal edges
isoexpr     string expression         Isosurface expression to plot
isolev      scalar{10}|vector         Number or vector of isosurface levels
isores      {[20 20 20]}              Isosurface interpolation resolution
sliceexpr   string expression         Slice expression to plot
slicex      [x-dir. middle of domain] Slice x-position coordinates
slicey      [y-dir. middle of domain] Slice y-position coordinates
slicez      [z-dir. middle of domain] Slice z-position coordinates
scatterexpr string expression         Scatter expression to plot
arrowexpr   3 string expressions      Arrow expressions to plot
arrowspacing vector {[10 10 10]}      Number of arrows in each direction
arrowcolor  {r}                       Arrow color
solnum      scalar {n_sols}           Solution number/time to plot
facecolor   {interp}                  Cell face style or fixed color
linestyle   {<tt>-</tt>}                     Cell edge line style
linewidth   {1}                       Line width
edgecolor   {k}                       Cell edge line color
boundary    off|{on}                  Show geometry boundary
alpha       {1}                       Transparency level
plotly      {0}                       Set plot parameters to support plotly export
title       string                    Plotly plot title
renderer    {empty string}|plotly     Render in Matlab/Octave or Plotly
parent      {gca}                     Axes handle to plot in
```

11.322 post/princse

PRINCSE Compute principal stresses and strains.

[SE_P] =

PRINCSE(SE_X, SE_Y, SE_Z, SE_XY, SE_YZ, SE_XZ, ICOMP) Computes principal stresses and strains (for the ICOMP component, default 1). In two dimensions only inputs SE_X, SE_Y, SE_Z, SE_XY, and ICOMP are required.

11.323 post/setaxes

SETAXES Apply plot axis settings.

11.324 post/slice3

SLICE3 Generate a triangular slice grid from tetrahedral grid.

[PS, TS, E] =

SLICE3(POS, P, T) Generates a triangular slice grid for the plane defined in POS, and the tetrahedral grid with vertices P and connectivities T. Returns vertices in PS and connectivities in TS for the corresponding grid. Optionally also computes and returns the boundary edges in E. (Applies Gaussian smoothing to inner slice nodes.)

11.325 postplot

POSTPLOT Postprocessing and visualization function.

[H] =

POSTPLOT(PROB, VARARGIN) Postprocessing function to plot and visualize FEATool problems. PROB is a valid finite element problem struct. Accepts the following property/value pairs and optionally returns an array H of plot object handles.

```
Property    Value/{Default}           1D 2D 3D Description
-------------------------------------------------------------------------------
surfexpr    string expression         x  x  x  Surface plot expression
surfhexpr   string expression            x     Expression for surface height plot
isoexpr     string expression            x  x  Contour/iso-surface plot expression
isolev      scalar{10}|vector            x  x  Number/vector of contour/iso-levels
isores      {[20 20 20]}                    x  Isosurface interpolation resolution
isocolor    {<tt>k</tt>}                       x     Contour/iso-line color
arrowexpr   n_sdim string expressions    x  x  Cell with arrow plot expressions
arrowcolor  {r}                          x  x  Arrow color
arrowspacing vector {[10 10 (10)]}       x  x  Number of arrows in each direction
sliceexpr   string expression               x  Slice plot expression
slicex      [x-dir. middle of domain]       x  Slice x-position coordinates
slicey      [y-dir. middle of domain]       x  Slice y-position coordinates
slicez      [z-dir. middle of domain]       x  Slice z-position coordinates
solnum      scalar {n_sols}           x  x  x  Solution number/time to plot
evalstyle   exact|{average}           x  x     Shape function evaluation type
axis        off|{on}                  x  x  x  Show axes
grid        on |{off}                 x  x  x  Show grid
view        {[-35 20]}                      x  3D view setting
axequal     off|{on}                  x  x  x  Axis equal setting
edges       on |{off}                       x  Show internal edges for surface plot
boundary    off|{on}                     x  x  Show geometry boundary
bbox        {0.05}                    x  x  x  Size of bounding box (0=off)
alpha       {1}                             x  Transparency level
facecolor   {interp}                     x  x  Cell face style or fixed color
```

```
linestyle   {1/3D:-}|{2D:none}        x  x  x  Line style
linewidth   {1}                       x  x     Line width
edgecolor   {k}                          x     Cell edge line color
color       {k}                       x        Line color
marker      {none}                    x        Marker style
markersize  {5}                       x        Marker size
markercolor {k}                       x        Marker color
cellabels   on|{off}                  x  x  x  Show cell numbers
nodelabels  on|{off}                  x  x  x  Show vertex/node numbers
fontsize    {axes default}            x  x  x  Font size used in text
colormap    {jet}                     x  x  x  Colormap
colorbar    {1D:off}|{2/3D:on}        x  x  x  Colorbar toggle
location    {eastoutside}             x  x  x  Colorbar location
selexpr     string expression         x  x  x  Expression to select cells to plot
selcells    {all}                     x  x  x  Index vector to cells to plot
selsubd     {all}                     x  x  x  Index vector to subdomains to plot
title       string                    x  x  x  Plotly plot title
renderer    {empty string}|plotly     x  x  x  Render in Matlab/Octave or Plotly
parent      {gca}                     x  x  x  Plot axes handle
setaxes     {on}|off                  x  x  x  Use axes modification settings
```

11.326 solvestat

SOLVESTAT Solve stationary problem.

[U] =

SOLVESTAT(PROB, VARARGIN) Solves the stationary problem described in the PROB finite element struct. Accepts the following property/value pairs.

```
Input       Value/{Default}              Description
-------------------------------------------------------------------------------
```

nsolve scalar/{1} Non-linear solution method

1 = Fixed point (Picard) iteration 2 = Newton method $<$0 = |nsolve|-Picard steps before Newton

jac scalar/struct {0} Newton Jacobian computation type

. - Direct assembly using symbolic jac. form and jac.coef struct contents 0 - Assembly by numerical differentiation minnit scalar/{0} Minimum number of non-linear iterations maxnit scalar/{25} Maximum number of non-linear iterations nlrlx scalar/string/{1.0} Relaxation

between non-linear iterations, scalar or string expr. ex 0.5∗(1+(it>5)) nlinasm logical/{ [0 0 0]} Force non-linear reassembly of A, f, bdrn toldef scalar/{1e-6} Stopping criteria for solution defects tolchg scalar/{1e-6} Stopping criteria for solution changes reldef logical/{0} Check relative defect changes relchg logical/{1} Check relative solution changes init u0|{expr}/{0} Initial value solution or experession solcomp {all dvars} Dep. variables/subdomains to solve for icub scalar/{2} Cubature rule/order used in assembly isymm scalar/{0} Symmetrize BCs if applicable waitbar scalar/{0} Show waitbar fid scalar/{1} File identifier for output ([]=no output)

See also

solvetime

11.327 solvetime

SOLVETIME Solve time-dependent problem.

[U, TLIST] =

SOLVETIME(PROB, VARARGIN) Solves the time-dependent problem described in the PROB finite element struct. Accepts the following property/value pairs.

Input	Value/{Default}	Description
tstep	scalar/{0.1}	Time step size (average for FS-scheme)
tmax	scalar/{1}	Maximum simulation time
tstop	scalar/{1e-6}	Stopping criterita (for solution changes in time)

ischeme scalar/{2} Time stepping scheme

1 = 1st order Backward Euler 2 = 2nd order Crank-Nicolson 3 = 2nd order Fractional-step-theta

imass scalar/{2} Mass matrix lumping

1 = Full mass matrix 2 = row sum lumping 3 = diagonal lumping 4 = HRZ diagonal lumping icub scalar/{2} Cubature rule/order used in assembly minnit scalar/{0} Minimum number of non-linear iterations maxnit scalar/{20} Maximum number of non-linear iterations

nstbwe scalar/{0} Number of initial forced BE steps nlrlx scalar/string/{1.0} Relaxation for non-linear iter., scalar or string expression, ex (1+(t_sim>5))/2 nlinasm logical/{ [0 0 0 0]} Force reassembly of M, A, f, bdrn toldef scalar/{1e-6} Stopping criteria for solution defects tolchg scalar/{1e-6} Stopping criteria for solution changes reldef logical/{0} Check relative defect changes relchg logical/{0} Check relative solution changes init u0|{expr}/{0} Initial value solution or experession solcomp {all dvars} Dep. variables/subdomains to solve for waitbar scalar/{0} Show waitbar fid scalar/{1} File identifier for output ([]=no output)

See also

solvestat

11.328 uninstall

UNINSTALL Remove the FEATool toolbox from the Octave/Matlab path.

UNINSTALL removes the FEATool toolbox from the Octave/Matlab paths.

11.329 util/conngrp

CONNGRP Group connected sets.

[IND_GROUPS, N_GROUPS] =

CONNGRP(C) Compute index to connected sets in connectivity array C. C may be symmetric or unsymmetric with rows corresponding to the grouping direction. Self referencing (isolated) entries will be omitted. Outputs are the total number of groups N_GROUPS and corresponding index array in IND_GROUPS of size 1 x number of C rows. A zero index indicates a removed only self referenced entry.

11.330 util/cstrcmp

CSTRCMP Compares cells of strings.

[S] =

CSTRCMP(C1, C2) Compares each element of C1 to the elements in C2. Returns the first

element in C1 found in C2. C1 can be a string or a cell of strings. If the flac F_NFFIRST is set and no strings were found S will return the first (default) entry in C1.

See also

strcmp, strncmp, strcmpi, strfind, deblank, regexp

11.331 util/deduplicate

DEDUPLICATE Remove duplicate rows or columns within tolerance.

[B, I, J] =

DEDUPLICATE(A, DIM, TOL, MODE) Removes duplicate rows or columns (according to DIM) from array A with tolerance TOL (default eps∗1e3). Positive TOL indicates relative tolerance, and negative absolue. Returns an array B and optionally index vectors I and J so that B = A(I,:) and A = B(J,:). (Retains the order of the unique rows/columns in A). Additionally MODE is a flag to indicate the use of unique built-in function (MODE = 0) or not (default).

See also

unique

11.332 util/dehtmlify

DEHTMLIFY Remove html clauses from string.

11.333 util/expandeqn

EXPANDEQN Expands an equation string expression.

[S] =

EXPANDEQN(S, C_SDIM, IDBG) Parses and expands an equation string expression S into terms. Additionally removes unnecessary parentheses. The optional cell array of space dimension names C_SDIM also parses parentheses with trailing derivative postfixes (.)_xi to [..]_xi. The optional IDBG flag turns on debugging output.

11.334 util/findparblock

FINDPARBLOCK Find parenthesis blocks.

[IP, SP] =

FINDPARBLOCK(S, I_EXCLUDE) Finds parenthesis blocks. Returns IP with index >0 for those characters in S contined in parentheses (including parentheses pairs). If the optional argument I_EXCLUDE is >0 then parenthesis with the preceeding functions and operators ^, /, sin, cos, tan, exp, log, log2, sqrt, and log10 will be excluded.

11.335 util/interpl

INTERPL Linear interpolation between points.

[PI] =

INTERPL(P, H) Applies linear interpolation with distance H between points in P.

11.336 util/mergeeqncoefs

MERGEEQNCOEFS Merge eqn struct coefficient fields.

[EQN] =

MERGEEQNCOEFS(EQN) Merge EQN struct coef fields which have the same form.

11.337 util/parsecoef

PARSECOEF Parse coefficients in problem struct.

[PROB] =

PARSECOEF(PROB, TOL) Parses and simplifies the eqn.m, a, f, and bdr.d and f coefficients. Includes expanding coefficients to all subdomains substituting expressions from the coef, const, expr, and vars fields. Also attempts to converts string coefficients to numeric and set them to zero if below tolerance TOL. If all coefficients in for form is zero it is returned empty instead. Furthermore, contracts all coefficient for a subdomain if all are identical.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
prob        struct                 Problem definition struct
```

```
tol         scalar                  Tolerance for zeroing coef (default eps*1e1)
                                                                                .
Output      Value/(Size)            Description
--------------------------------------------------------------------------------
prob        struct                  Processed problem definition struct
```

11.338 util/parseeqn

PARSEEQN Parse string equations and return eqn struct.

[EQN] =

PARSEEQN(CEQN, CDVAR, CSDIM) Parse string equation(s) and return weak finite element equation struct form and coefficients.

```
Input        Value/Type             Description
-------------------------------------------------------------------------------
ceqn         cell array/string      Equation string or cell array of strings
cdvar        cell array/string      Dependent variable name(s)
csdim        cell array/string      Space dimension coordinate name(s)
```

String quations supports the following syntax

```
u \form#224 = single quote)
u      - dependent variable u (explicit)
u_t    - dependent variable u (implicit)
ux     - space derivative for u in x-direction (explicit, rhs)
u_x    - space derivative for u in x-direction (implicit)
ux_t   - space derivative for u in x-direction (implicit)
ux_x   - 2nd space derivative for u in x-direction
(.)_t  - Multiplication of expression with test function
(.)_x  - Multiplication of expr. with test function in x-direction
(.)_tx - Multiplication of expr. with test function in x-direction
```

an underscore indicates multiplication with a test function, t, operating on the preceding expression. For expressions in parentheses with multiple dependent variable factors the operation is applied to the first from the reversed direction, for example (2∗uy+u∗ux)_x will be expanded to 2∗uy_x+u∗ux_x. Application of derivatives _x to constants, variables, or first of dependent variables will also change the equation sign accounting partial integration in the weak fem formulations. In addition expressions can be built up with standard Matlab commands, functions, and operators such + - ∗ / sqrt() log() etc.

Example

1) Poisson's equation in 1D

```
seqn = <tt>da*u</tt><tt> - (D*ux)_x = f</tt>;
eqn  = parseeqn( seqn, <tt>u</tt>, <tt>x</tt> );

Gives, eqn: the mass matrix, in this case scaled by da

eqn.form{:} >> [1;1]
eqn.coef{:} >> da

eqn.a: implicit bilinear form, here diffusion matrix scaled by D

eqn.a.form{:} >> [2;2]
eqn.a.coef{:} >> D

eqn.f: linear right hand side source term

eqn.f.form{:} >> 1
eqn.f.coef{:} >> f
```

See also

expandeqn, assemblea, assemblef

11.339 util/parseexpr

PARSEEXPR Parses a string expression.

[S_EXPR, CSUBEXPR] =

PARSEEXPR(S_EXPR, S_SUBEXPR, C_ALLOWEXPR) Parses the string expression in S_EXPR and returns a cell array CSUBEXPR with variable expressions to process. Also returns a modified expression S_EXPR which has the sub variable names substituted with S_SUBEXPR(:,I) where I is the index in CSUBEXPR to the corresponding variable. This enables a call to EVAL(S_EXPR) after the expressions in CSUBEXPR has been computed and copied to the S_SUBEXPR array. The optional array C_ALLOWEXPR allows strings as subexpressions (which function names would otherwise be removed).

See also

symvarbe

11.340 util/parseopt

PARSEOPT Parse options.

[GOT, VAL] =

PARSEOPT(COPTDEF, VARARGIN) parses the property/value pairs in VARARGIN, and adds the default options in the second column of COPTDEF. The first column of COPTDEF contains strings of the field names. Returns the got/val structs.

11.341 util/parsephys

PARSEPHYS Check and parse physics modes.

[SOUT] =

PARSEPHYS(SIN, F_ADD) Checks and parses the physics modes in the given struct SIN.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
sin         struct                 Problem definition or physics mode struct
f_add       logical                Flag to expand and add physics mode
                                   data to problem struct fields
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
sout        struct                 Parsed problem or physics mode struct
```

See also

parseprob, parseprob0

11.342 util/parseprob

PARSEPROB Check and parse a problem struct.

[PROB] =
PARSEPROB(PROB) Checks and parses the PROB struct on the high (equation) formulation level.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
prob        struct                 Problem definition struct
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
prob        struct                 Parsed problem definition struct
```

See also

parseprob0, parsephys

11.343 util/parseprob0

PARSEPROB0 Low level check and parsing of a problem struct.
[PROB] =
PARSEPROB0(PROB) Checks and parses the PROB struct on the low (weak/form) level.

```
Input       Value/(Size)           Description
-------------------------------------------------------------------------------
prob        struct                 Problem definition struct
                                                                              .
Output      Value/(Size)           Description
-------------------------------------------------------------------------------
prob        struct                 Parsed problem definition struct
```

See also

parseprob, parsephys

11.344 util/recursive_termsplit

RECURSIVE_TERMSPLIT Splits a string expression recursively into terms.

[SIGNS, TERMS, DSIGNS] =

RECURSIVE_TERMSPLIT(S) Splits the string expression S into a cell array of TERMS with preceding SIGNS. DSIGNS optionally indicates the signs with doubles (1 or -1).

11.345 util/remove_duplicate_rows

REMOVE_DUPLICATE_ROWS Removes duplicate rows.

[P_OUT] =

REMOVE_DUPLICATE_ROWS(P, TOL) Removes duplicate rows from array P within tolerance TOL (default eps∗1e3). (Keeps the first found duplicate rows and sorts the output so as to preserve the order as much as possible). Optionally returns an index vector IND to the original rows in P so that P_OUT = P(IND,:).

11.346 util/remove_parentheses

REMOVE_PARENTHESES Remove outer parentheses from string expression.

[SGN, S] =

REMOVE_PARENTHESES(S) Finds and removes outer parentheses from string expression S. The resulting sign will be returned in SGN as a scalar (1 or -1). If nothing could be done SGN will return 0.

11.347 util/rmkdir

RMKDIR Recursively create directories.

11.348 util/set_geomlib

SET_GEOMLIB Set geomlib version.

SET_GEOMLIB(VER) Sets geometry library path to use. VER=1 sets the old v1.7 geometry library (geom1), VER=2 sets the new geometry library (geom2).

11.349 util/stfldcheck

STFLDCHECK Traverse the struct ST0 and checks if all fields in C exist.

[F] =

STRFLDCHECK(C, ST0, F_ISEMPTYCHK) Traverses down the ST0 struct accordint to the fieldnames in C and checks if they are present. Additionally checks if the field is empty if F_ISEMPTYCHK is set.

11.350 util/strconv

STRCONV Convert string into numeric and string data.

[DATA] =

STRCONV(S, FORMAT) Splits and converts the string S into numeric and string data accoring to FORMAT. FORMAT is either a space separated string or cell array with format specifiers, s for string and anything else for numeric data. Output is a cell array DATA with converted results according to FORMAT. Line breaks will be removed. If FORMAT is empty or not present S will be split into default numerical data and strings.

See also

sscanf

11.351 util/stringify

STRINGIFY Convert input to string.

[STR] =

STRINGIFY(ARG, EXTRACT_CONTENTS, MERGE_LINES, STRUCT_BASE_NAME) Converts the data in ARG to a string so that it can be executed as a command and reprodce the input. EXTRACT_CONTENTS indicate if cells should be expanded (default 0). MERGE_LINES toggles merging of rows to a single row (default 0). STRUCT_BASE_NAME is a string placeholder for the head of a struct (default simply struct).

11.352 util/symvarbe

SYMVARBE Determine the symbolic variables in an expression.

[VARS, I_START, I_END] =

SYMVARBE(S) searches the string S for identifiers other than `pi`, `inf`, `nan`, `eps` and common functions. The variables are returned as a cell array of strings with start and end index given in I_START and I_END respectively. If no such variable exists, the empty cell array {} is returned.

Example

```
symvarbe( 'cos(pi*x - beta1)' )  returns  {'x' 'beta1'} [8 12] [8 16]
```

11.353 util/termsplit

TERMSPLIT Splits string into terms.

[CSIGN, CTERM, B, E] =

TERMSPLIT(S) Splits string expression in S into terms in CTERM with sign CSIGN. B and E indicates start and end position in S of the terms.

11.354 util/timerfun

TIMERFUN Create single timer function.

[H] =

TIMERFUN(CBF, DELAY) Creates a single timer function with persistent handle H. If a timer function is not already present a new object is created with the callback CBF which is called every DELAY seconds (CBF can be a cell array of a function handle and following input arguments). A call with an existing timer object stops and deletes it.

See also

timer

11.355 util/urlget

URLGET Get/fetch file from url.

[F, STATUS] =

URLGET(URL, FILENAME) Get/fetch file from URL and save to FILENAME. Returns the path to the file in F, and STATUS 1 if successful.

11.356 util/uunique

UUNIQUE Unsorted set unique.

[B, I, J] =

UUNIQUE(A) for the vector or row array A returns the same values in the same order as in A without repetitions. I is an index vector so that A = B(I), and J with B = A(J).

See also

unique