

OpenCMISS-iron examples and tests used by OpenCMISS developers at University of Stuttgart, Germany

Christian Bleiler*, Dr.-Ing. Nehzat Emamy[†]
Andreas Hessenthaler*, Thomas Klotz*,
Aaron Krämer[‡], Benjamin Maier[†], Sergio Morales*,
Mylena Mordhorst*, Harry Saini*

August 2, 2017
16:53

CONTENTS

1	Test summary	6
1.1	Details	6
2	Introduction	8
2.1	Cogui files for cmgui-2.9	8
2.2	Variations to consider	8
2.3	Folder structure	9
3	Progress	10
3.1	Equations to test	10
3.2	Setting up a new test	10
3.3	Long-term goals	11
4	Diffusion equation	12
4.1	Equation in general form	12
4.2	Example-0001 [VALIDATED]	13
4.2.1	Mathematical model - 2D	13
4.2.2	Mathematical model - 3D	13
4.2.3	Computational model	13
4.2.4	Result summary	14
4.3	Example-0001-u [PLAUSIBLE]	17
4.3.1	Mathematical model - 2D	17
4.3.2	Mathematical model - 3D	17
4.3.3	Computational model	17

* Institute of Applied Mechanics (CE), University of Stuttgart, Pfaffenwaldring 7, 70569 Stuttgart, Germany

† Institute for Parallel and Distributed Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany

‡ Lehrstuhl Mathematische Methoden für komplexe Simulation der Naturwissenschaft und Technik, University of Stuttgart, Allmandring 5b, 70569 Stuttgart, Germany

4.3.4	Result summary	18
4.4	Example-0002 [VALIDATED]	21
4.4.1	Mathematical model - 2D	21
4.4.2	Mathematical model - 3D	21
4.4.3	Computational model	21
4.4.4	Result summary	22
4.5	Example-0003 [COMPILES]	25
4.5.1	Mathematical model - 2D	25
4.5.2	Mathematical model - 3D	25
4.5.3	Computational model	25
4.5.4	Result summary	26
4.6	Example-0004 [VALIDATED]	29
4.6.1	Mathematical model - 2D	29
4.6.2	Computational model	29
4.6.3	Result summary	29
4.7	Example-0005 [VALIDATED]	31
4.7.1	Mathematical model - 2D	31
4.7.2	Mathematical model - 3D	31
4.7.3	Computational model	31
4.7.4	Result summary	32
4.8	Example-0011 [VALIDATED]	35
4.8.1	Mathematical model - 2D	35
4.8.2	Mathematical model - 3D	35
4.8.3	Computational model	35
4.8.4	Result summary	36
5	Linear elasticity	39
5.1	Equation in general form	39
5.2	Example-0101 [PLAUSIBLE]	40
5.2.1	Mathematical model	40
5.2.2	Computational model	40
5.2.3	Results	41
5.2.4	Validation	42
5.3	Example-0102 [PLAUSIBLE]	44
5.3.1	Mathematical model	44
5.3.2	Computational model	44
5.3.3	Results	45
5.3.4	Validation	46
5.4	Example-0111 [PLAUSIBLE]	48
5.4.1	Mathematical model	48
5.4.2	Computational model	48
5.4.3	Results	49
5.4.4	Validation	50
5.5	Example-0112 [PLAUSIBLE]	52
5.5.1	Mathematical model	52
5.5.2	Computational model	52
5.5.3	Results	53
5.5.4	Validation	54
6	Finite elasticity	56
7	Navier-Stokes flow	57
7.1	Equation in general form	57
7.2	Example-0302-u [COMPILES]	58
7.2.1	Mathematical model - 2D	58
7.2.2	Mathematical model - 3D	59

7.2.3	Computational model	59
7.2.4	Result summary	60
8	Monodomain	61
8.1	Example-0401 [PLAUSIBLE]	62
8.1.1	Mathematical model	62
8.1.2	Computational model	62
8.1.3	Results	63
8.1.4	Validation	63
8.2	Example-0402 [PLAUSIBLE]	66
8.2.1	Mathematical model	66
8.2.2	Computational model	66
8.2.3	Results	67
8.2.4	Validation	67
8.3	Example-0404-c [PLAUSIBLE]	69
8.3.1	Mathematical model	69
8.3.2	Computational model	69
8.3.3	Results	70
8.3.4	Validation	70
9	CellML model	73

LIST OF FIGURES

Figure 1	2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].	14
Figure 2	2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].	15
Figure 3	3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].	15
Figure 4	3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].	16
Figure 5	2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].	19
Figure 6	2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].	19
Figure 7	3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].	20
Figure 8	3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].	20
Figure 9	2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].	22
Figure 10	2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].	23
Figure 11	3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].	23
Figure 12	3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].	24
Figure 13	2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].	27
Figure 14	2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].	27
Figure 15	3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].	28

Figure 16	3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].	28
Figure 17	2D results, iron reference w/ command line arguments [8 4 0 2 0].	30
Figure 18	2D results, current run w/ command line arguments [8 4 0 2 0].	30
Figure 19	2D geometry and mesh.	32
Figure 20	2D results, iron reference w/ command line arguments [2 2 0].	32
Figure 21	2D results, current run w/ command line arguments [2 2 0].	33
Figure 22	3D geometry (unit cube) and mesh.	33
Figure 23	3D results, iron reference w/ command line arguments [3 2 0].	34
Figure 24	3D results, current run w/ command line arguments [3 2 0].	34
Figure 25	2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0 1 1].	37
Figure 26	2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0 1 1].	37
Figure 27	3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0 1 1].	38
Figure 28	3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0 1 1].	38
Figure 29	Results, iron 2D fine mesh.	41
Figure 30	Results, iron 3D fine mesh.	41
Figure 31	Results, Abaqus 2D fine mesh.	42
Figure 32	Results, abaqus 3D fine mesh.	43
Figure 33	Results, iron 2D fine mesh.	45
Figure 34	Results, iron 3D fine mesh.	45
Figure 35	Results, Abaqus 2D fine mesh.	46
Figure 36	Results, abaqus 3D fine mesh.	47
Figure 37	Results, iron 2D fine mesh.	49
Figure 38	Results, iron 3D fine mesh.	49
Figure 39	Results, Abaqus 2D fine mesh.	50
Figure 40	Results, abaqus 3D fine mesh.	51
Figure 41	Results, iron 2D fine mesh.	53
Figure 42	Results, iron 3D fine mesh.	53
Figure 43	Results, Abaqus 2D fine mesh.	54
Figure 44	Results, abaqus 3D fine mesh.	55
Figure 45	Results movie, 24×24 elements (only works in certain pdf viewers, e.g. Adobe Acrobat Reader)	63
Figure 46	Results, 10×10 elements, $t = 200$	64
Figure 47	Results, 24×24 elements, $t = 200$	64
Figure 48	Results, 24×24 elements, $t = 500$	65
Figure 49	Results, 24×24 elements, $t = 200$	67
Figure 50	Results, 10×10 elements	68
Figure 51	V_m for time $t = 1.0$, different time step widths $dt \in \{0.01, 0.005, 0.001, 0.0005, 0.00025\}$	70
Figure 52	Error at $t = 1.0$ for different time steps widths. The slope (=experimental order of convergence) should be around 1.	71
Figure 53	V_m for time $t = 3.0$, different time step widths $dt \in \{0.01, 0.005, 0.001, 0.0005, 0.00025\}$	71

Figure 54	Error at $t = 3.0$ for different time steps widths. The slope (=experimental order of convergence) should be around 1.	72
-----------	--	----

LIST OF TABLES

Table 1	Quantitative error between Abaqus 2017 and iron simulations for linear elastic uniaxial extensions	43
Table 2	Quantitative error between Abaqus 2017 and iron simulations for linear elastic shear	47
Table 3	Quantitative error between Abaqus 2017 and iron simulations for linear elastic uniaxial extensions	51
Table 4	Quantitative error between Abaqus 2017 and iron simulations for linear elastic shear	55

1 TEST SUMMARY

Passed tests: 192 / 230

1.1 Details

Content of: example-0001/results/failed.tests

No failed tests.

Content of: example-0001-u/results/failed.tests

current_run/l2x1x0_n8x4x0_i8_s0/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i8_s1/Example.part0.exnode

| CHeart - Iron | $_2$ = 0.05804
| CHeart - Iron | $_2$ = 0.05804

Content of: example-0002/results/failed.tests

No failed tests.

Content of: example-0003/results/failed.tests

Failed tests:

current_run/l2x1x0_n2x1x0_i1_s0/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i1_s0/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i1_s0/Example.part0.exnode
current_run/l2x1x0_n2x1x0_i2_s0/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i2_s0/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i2_s0/Example.part0.exnode
current_run/l2x1x0_n2x1x0_i1_s1/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i1_s1/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i1_s1/Example.part0.exnode
current_run/l2x1x0_n2x1x0_i2_s1/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i2_s1/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i2_s1/Example.part0.exnode
current_run/l2x1x0_n2x1x0_i1_s0/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i1_s0/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i1_s0/Example.part0.exnode
current_run/l2x1x1_n2x1x1_i1_s0/Example.part0.exnode
current_run/l2x1x1_n4x2x2_i1_s0/Example.part0.exnode
current_run/l2x1x1_n8x4x4_i1_s0/Example.part0.exnode
current_run/l2x1x1_n2x1x1_i2_s0/Example.part0.exnode
current_run/l2x1x1_n4x2x2_i2_s0/Example.part0.exnode
current_run/l2x1x1_n8x4x4_i2_s0/Example.part0.exnode
current_run/l2x1x1_n2x1x1_i1_s1/Example.part0.exnode
current_run/l2x1x1_n4x2x2_i1_s1/Example.part0.exnode
current_run/l2x1x1_n8x4x4_i1_s1/Example.part0.exnode
current_run/l2x1x1_n2x1x1_i2_s1/Example.part0.exnode
current_run/l2x1x1_n4x2x2_i2_s1/Example.part0.exnode
current_run/l2x1x1_n8x4x4_i2_s1/Example.part0.exnode

| CHeart - Iron | $_2$ = 44.2627
| CHeart - Iron | $_2$ = 37.2770
| CHeart - Iron | $_2$ = 32.2165
| CHeart - Iron | $_2$ = 27.3358
| CHeart - Iron | $_2$ = 22.1869
| CHeart - Iron | $_2$ = 19.7449
| CHeart - Iron | $_2$ = 44.2627
| CHeart - Iron | $_2$ = 37.2770
| CHeart - Iron | $_2$ = 32.2165
| CHeart - Iron | $_2$ = 27.3358
| CHeart - Iron | $_2$ = 22.1869
| CHeart - Iron | $_2$ = 19.7449
| CHeart - Iron | $_2$ = 124.749
| CHeart - Iron | $_2$ = 128.672
| CHeart - Iron | $_2$ = 143.606
| CHeart - Iron | $_2$ = 94.2619
| CHeart - Iron | $_2$ = 98.7606
| CHeart - Iron | $_2$ = 118.047
| CHeart - Iron | $_2$ = 124.749
| CHeart - Iron | $_2$ = 128.672
| CHeart - Iron | $_2$ = 143.606
| CHeart - Iron | $_2$ = 94.2619
| CHeart - Iron | $_2$ = 98.7606
| CHeart - Iron | $_2$ = 118.047

Content of: example-0004/results/failed.tests

No failed tests.

Content of: example-0005/results/failed.tests

No failed tests.

Content of: example-0011/results/failed.tests

No failed tests.

Content of: example-0302-u/results/failed.tests

All tests failed.

Content of: example-0401/results/failed.tests
No failed tests.

Content of: example-0402/results/failed.tests
No failed tests.

2 INTRODUCTION

This document contains information about examples used for testing *OpenCMISS-iron*. Read: How-to¹ and [1].

2.1 Cogui files for cmgui-2.9

2.2 Variations to consider

- Geometry and topology
 - 1D, 2D, 3D
 - Length, width, height
 - Number of elements
 - Interpolation order
 - Generated or user meshes
 - quad/hex or tri/tet meshes
- Initial conditions
- Load cases
 - Dirichlet BC
 - Neumann BC
 - Volume force
 - Mix of previous items
- Sources, sinks
- Time dependence
 - Static
 - Quasi-static
 - Dynamic
- Material laws
 - Linear
 - Nonlinear (Mooney-Rivlin, Neo-Hookean, Ogden, etc.)
 - Active (Stress, strain)
- Material parameters, anisotropy
- Solver
 - Direct
 - Iterative
- Test cases
 - Numerical reference data
 - Analytical solution
- A mix of previous items

¹ <https://bitbucket.org/hessenthaler/opencmiss-howto>

2.3 Folder structure

TBD..

3 PROGRESS

People working on setting up tests in alphabetical order (surnames) with initials:

- CB : Christian Bleiler
- NE : Dr.-Ing. Nehzat Emamy
- AH : Andreas Hessenthaler
- TK : Thomas Klotz
- AK : Aaron Krämer
- BM : Benjamin Maier
- SM : Sergio Morales
- MM : Mylena Mordhorst
- HS : Harry Saini

3.1 Equations to test

Test single-physics problems before multi-physics problems!

- Diffusion equation (Laplace, Poisson, Generalized Laplace, ALE Diffusion, etc.)
- Linear elasticity equation (compressible and incompressible)
- Finite elasticity equation (compressible and incompressible Mooney-Rivlin, etc.)
- Navier-Stokes equation (ALE, Stokes, etc.)
- Monodomain equation
- CellML models
- Skeletal muscle models
- Fluid-structure interaction
- etc.

3.2 Setting up a new test

Use the following guideline to set up a new test:

1. Check if it is already there
2. Talk to other developers
3. Create a new subfolder examples/example-xxxx
4. Document the setup (computational domain, etc.) in examples/example-xxxx/doc/example.tex
5. Set up example with all parameters as command line arguments, see Section [2.2](#)

6. Set up reference results (CHeart, Abaqus, analytical solution, etc.)
7. Set up script to run all tests in your example directory
8. Set up script to perform comparison between iron results and reference results
9. Set up visualization scripts
10. Compile, run, test, visualize your example
11. Compile, run, test, visualize all examples

For each example, progress is documented in the respective section titles with the following **TAG**:

- **DOCUMENTED**: finish the documentation of the example (spatial domain, number of time steps, boundary conditions, etc.)
- **COMPILES**: example compiles (for default parameters)
- **RUNS**: example runs (for default parameters)
- **CONVERGES**: no convergence issues (for default parameters, results not plausible)
- **PLAUSIBLE**: results look sensible (for default parameters)
- **VALIDATED**: for all parameter sets it gives the correct results as compared to CHeart/Abaqus/analytical solution (includes visualization scripts, run scripts, comparison scripts, documentation!, ...)

Move all tags **CONVERGE**, **PLAUSIBLE** to **VALIDATED**.

Next steps include:

- Everybody runs everything!
- Meeting with Oliver
- Meeting with Auckland

3.3 Long-term goals

- Different testing targets
 - SMALL : small, fast tests
 - BIG : same as before; further, bigger and more complex geometries, convergence analysis
 - PARALLEL : same as before but in parallel
- Add more examples/those which were on the agenda but not started
- Jenkins continuous testing, integration and deployment
 - test SMALL/BIG/PARALLEL targets
 - integrate with GitHub (pull-requests triggers Jenkins, merge on success)

4 DIFFUSION EQUATION

4.1 Equation in general form

The governing equation is,

$$\partial_t u + \nabla \cdot [\sigma \nabla u] = f, \quad (1)$$

with conductivity tensor σ . The conductivity tensor is,

- defined in material coordinates (fibre direction),
- diagonal,
- defined per element.

4.2 Example-0001 [VALIDATED]

Example uses generated regular meshes and solves a static problem, i.e., applies the boundary conditions in one step.

4.2.1 Mathematical model - 2D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1], \quad (2)$$

with boundary conditions

$$u = 0 \quad x = y = 0, \quad (3)$$

$$u = 1 \quad x = 2, y = 1. \quad (4)$$

No material parameters to specify.

4.2.2 Mathematical model - 3D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1] \times [0, 1], \quad (5)$$

with boundary conditions

$$u = 0 \quad x = y = z = 0, \quad (6)$$

$$u = 1 \quad x = 2, y = z = 1. \quad (7)$$

No material parameters to specify.

4.2.3 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

- Commandline arguments for tests are:

2.0 1.0 0.0 2 1 0 1 0

2.0 1.0 0.0 4 2 0 1 0

2.0 1.0 0.0 8 4 0 1 0

2.0 1.0 0.0 2 1 0 2 0

2.0 1.0 0.0 4 2 0 2 0

2.0 1.0 0.0 8 4 0 2 0

2.0 1.0 0.0 2 1 0 1 1

2.0 1.0 0.0 4 2 0 1 1

```

2.0 1.0 0.0 8 4 0 1 1
2.0 1.0 0.0 2 1 0 2 1
2.0 1.0 0.0 4 2 0 2 1
2.0 1.0 0.0 8 4 0 2 1
2.0 1.0 1.0 2 1 1 1 0
2.0 1.0 1.0 4 2 2 1 0
2.0 1.0 1.0 8 4 4 1 0
2.0 1.0 1.0 2 1 1 2 0
2.0 1.0 1.0 4 2 2 2 0
2.0 1.0 1.0 8 4 4 2 0
2.0 1.0 1.0 2 1 1 1 1
2.0 1.0 1.0 4 2 2 1 1
2.0 1.0 1.0 8 4 4 1 1
2.0 1.0 1.0 2 1 1 2 1
2.0 1.0 1.0 4 2 2 2 1
2.0 1.0 1.0 8 4 4 2 1

```

4.2.4 Result summary

We use CHeart rev. 6292 to produce numerical reference solutions.

Passed tests: 24 / 24

No failed tests.

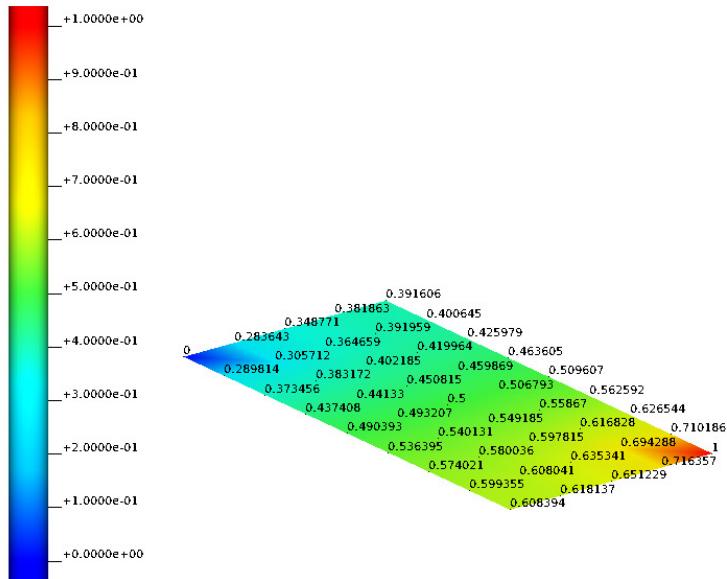


Figure 1: 2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].

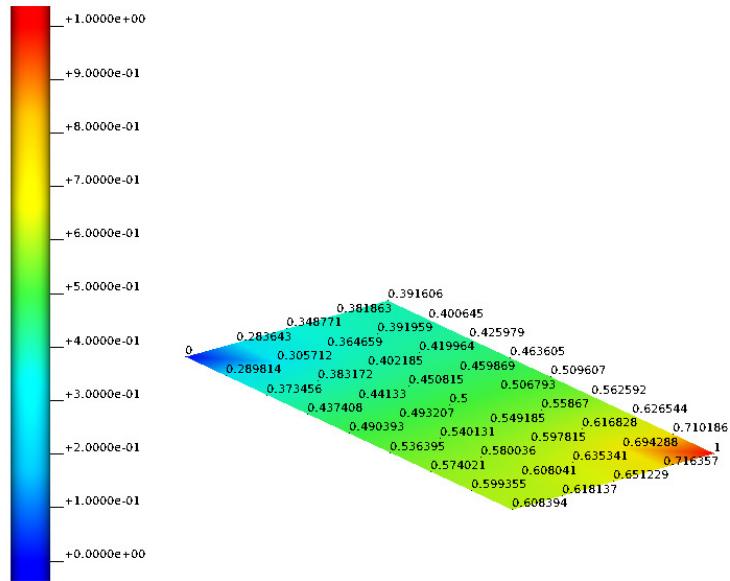


Figure 2: 2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].

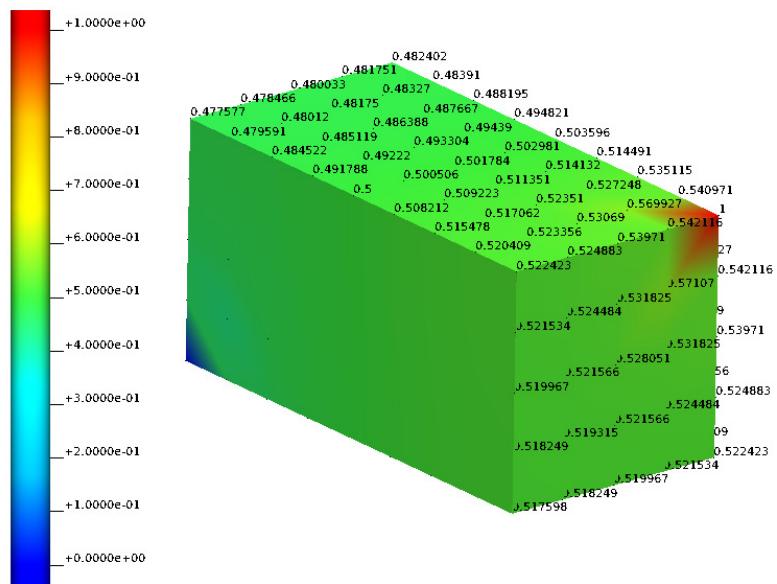


Figure 3: 3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].

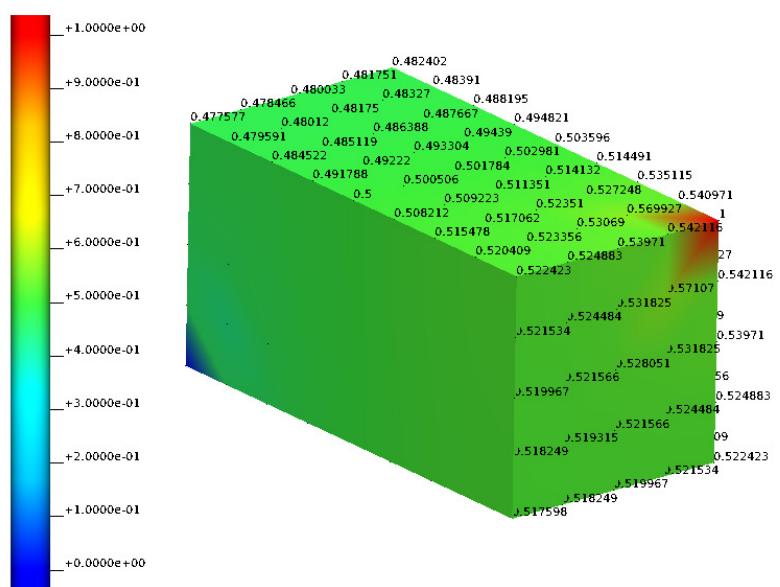


Figure 4: 3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].

4.3 Example-0001-u [PLAUSIBLE]

Example uses user-defined regular meshes in CHearm mesh format and solves a static problem, i.e., applies the boundary conditions in one step.

Issues: Interpolation type 8 (quadratic simplex) gives significantly different results compared to CHearm whereas all other simplex mesh results correspond rather well..

4.3.1 Mathematical model - 2D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1], \quad (8)$$

with boundary conditions

$$u = 0 \quad x = y = 0, \quad (9)$$

$$u = 1 \quad x = 2, y = 1. \quad (10)$$

No material parameters to specify.

4.3.2 Mathematical model - 3D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1] \times [0, 1], \quad (11)$$

with boundary conditions

$$u = 0 \quad x = y = z = 0, \quad (12)$$

$$u = 1 \quad x = 2, y = z = 1. \quad (13)$$

No material parameters to specify.

4.3.3 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

- Commandline arguments for tests are:

2.0 1.0 0.0 2 1 0 1 0

2.0 1.0 0.0 4 2 0 1 0

2.0 1.0 0.0 8 4 0 1 0

2.0 1.0 0.0 2 1 0 2 0

```

2.0 1.0 0.0 4 2 0 2 0
2.0 1.0 0.0 8 4 0 2 0
2.0 1.0 0.0 8 4 0 7 0
2.0 1.0 0.0 8 4 0 8 0
2.0 1.0 0.0 2 1 0 1 1
2.0 1.0 0.0 4 2 0 1 1
2.0 1.0 0.0 8 4 0 1 1
2.0 1.0 0.0 2 1 0 2 1
2.0 1.0 0.0 4 2 0 2 1
2.0 1.0 0.0 8 4 0 2 1
2.0 1.0 0.0 8 4 0 7 1
2.0 1.0 0.0 8 4 0 8 1
2.0 1.0 1.0 2 1 1 1 0
2.0 1.0 1.0 4 2 2 1 0
2.0 1.0 1.0 8 4 4 1 0
2.0 1.0 1.0 2 1 1 2 0
2.0 1.0 1.0 4 2 2 2 0
2.0 1.0 1.0 8 4 4 2 0
2.0 1.0 1.0 8 4 4 7 0
2.0 1.0 1.0 8 4 4 8 0
2.0 1.0 1.0 2 1 1 1 1
2.0 1.0 1.0 4 2 2 1 1
2.0 1.0 1.0 8 4 4 1 1
2.0 1.0 1.0 2 1 1 2 1
2.0 1.0 1.0 4 2 2 2 1
2.0 1.0 1.0 8 4 4 2 1
2.0 1.0 1.0 8 4 4 7 1
2.0 1.0 1.0 8 4 4 8 1

```

- Note: Binary uses command line arguments to search for the relevant mesh files.

4.3.4 Result summary

We use CHeart rev. 6292 to produce numerical reference solutions.

Passed tests: 30 / 32

current_run/l2x1x0_n8x4x0_i8_s0/Example.part0.exnode	CHeart	- Iron	_2 = 0.05804
current_run/l2x1x0_n8x4x0_i8_s1/Example.part0.exnode	CHeart	- Iron	_2 = 0.05804

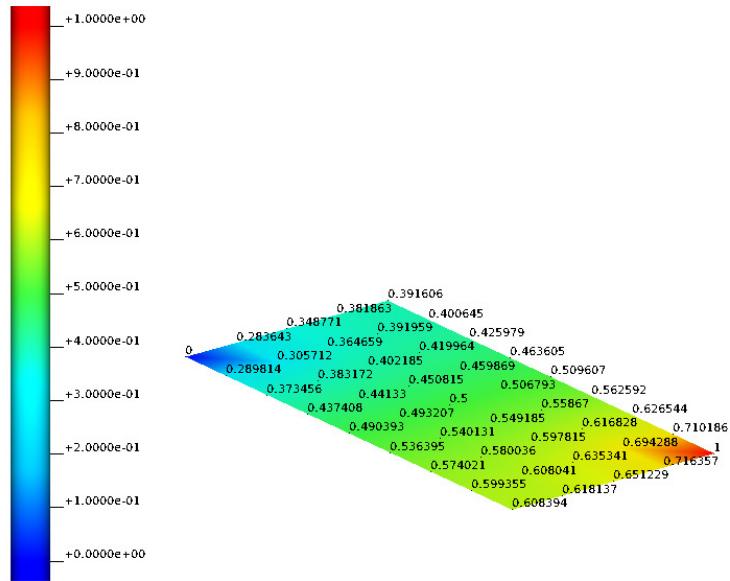


Figure 5: 2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].

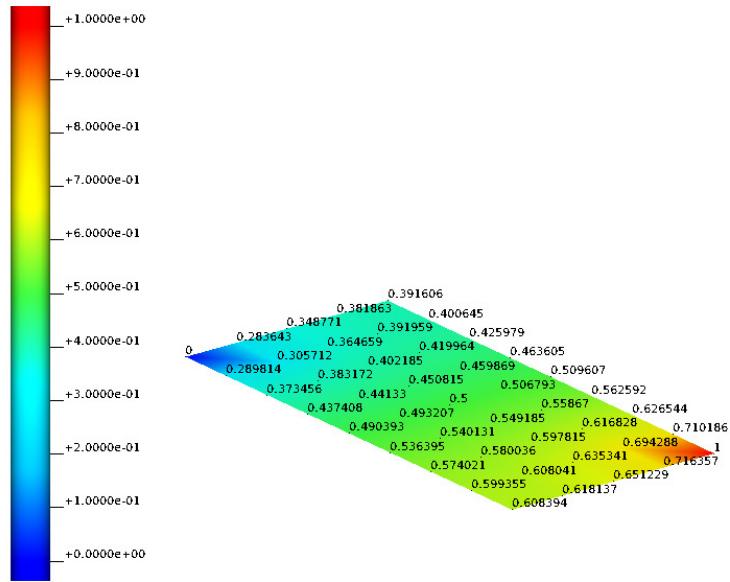


Figure 6: 2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].

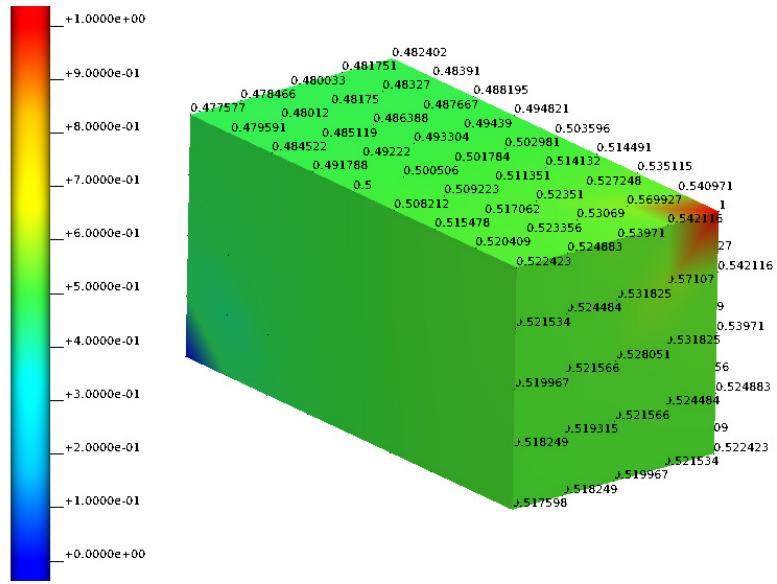


Figure 7: 3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].

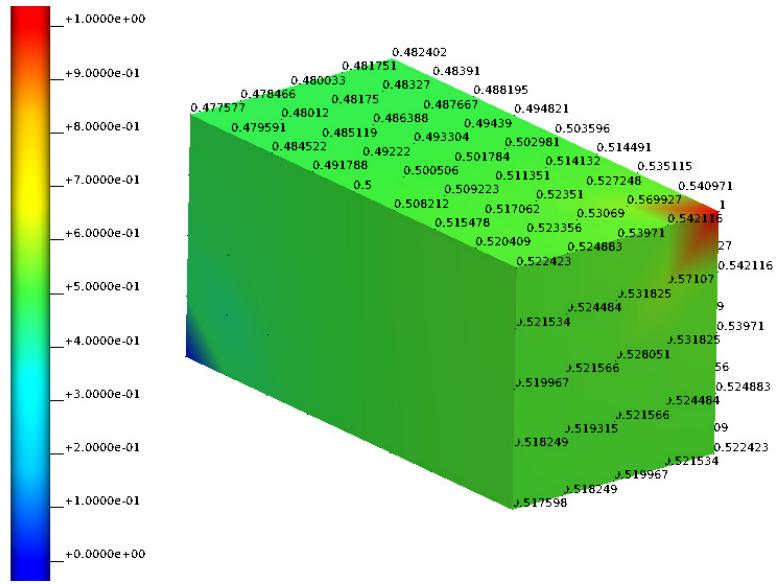


Figure 8: 3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].

4.4 Example-0002 [VALIDATED]

Example uses generated regular meshes and solves a static problem, i.e., applies the boundary conditions in one step.

4.4.1 Mathematical model - 2D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1], \quad (14)$$

with boundary conditions

$$u = 15y \quad x = 0, \quad (15)$$

$$u = 25 - 18y \quad x = 2. \quad (16)$$

No material parameters to specify.

4.4.2 Mathematical model - 3D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1] \times [0, 1], \quad (17)$$

with boundary conditions

$$u = 15y \quad x = 0, \quad (18)$$

$$u = 25 - 18y \quad x = 2. \quad (19)$$

No material parameters to specify.

4.4.3 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

- Commandline arguments for tests are:

2.0 1.0 0.0 2 1 0 1 0

2.0 1.0 0.0 4 2 0 1 0

2.0 1.0 0.0 8 4 0 1 0

2.0 1.0 0.0 2 1 0 2 0

2.0 1.0 0.0 4 2 0 2 0

2.0 1.0 0.0 8 4 0 2 0

2.0 1.0 0.0 2 1 0 1 1

2.0 1.0 0.0 4 2 0 1 1

```

2.0 1.0 0.0 8 4 0 1 1
2.0 1.0 0.0 2 1 0 2 1
2.0 1.0 0.0 4 2 0 2 1
2.0 1.0 0.0 8 4 0 2 1
2.0 1.0 1.0 2 1 1 1 0
2.0 1.0 1.0 4 2 2 1 0
2.0 1.0 1.0 8 4 4 1 0
2.0 1.0 1.0 2 1 1 2 0
2.0 1.0 1.0 4 2 2 2 0
2.0 1.0 1.0 8 4 4 2 0
2.0 1.0 1.0 2 1 1 1 1
2.0 1.0 1.0 4 2 2 1 1
2.0 1.0 1.0 8 4 4 1 1
2.0 1.0 1.0 2 1 1 2 1
2.0 1.0 1.0 4 2 2 2 1
2.0 1.0 1.0 8 4 4 2 1

```

4.4.4 Result summary

We use CHeart rev. 6292 to produce numerical reference solutions.

Passed tests: 24 / 24

No failed tests.

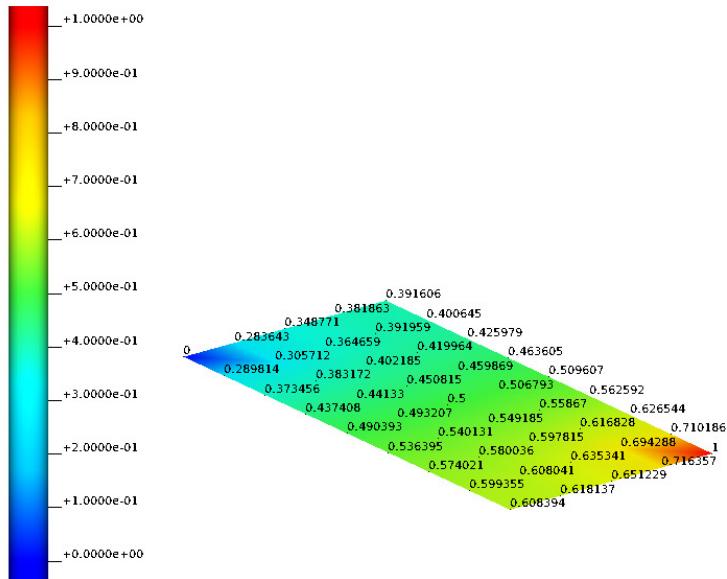


Figure 9: 2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].

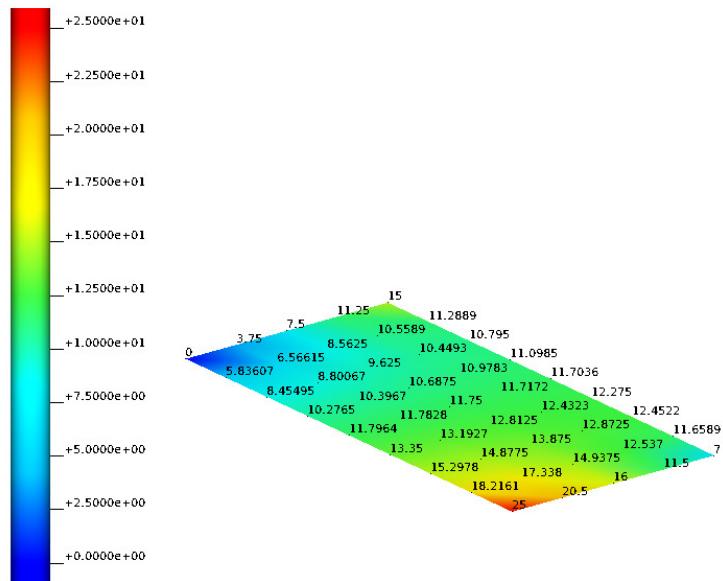


Figure 10: 2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].

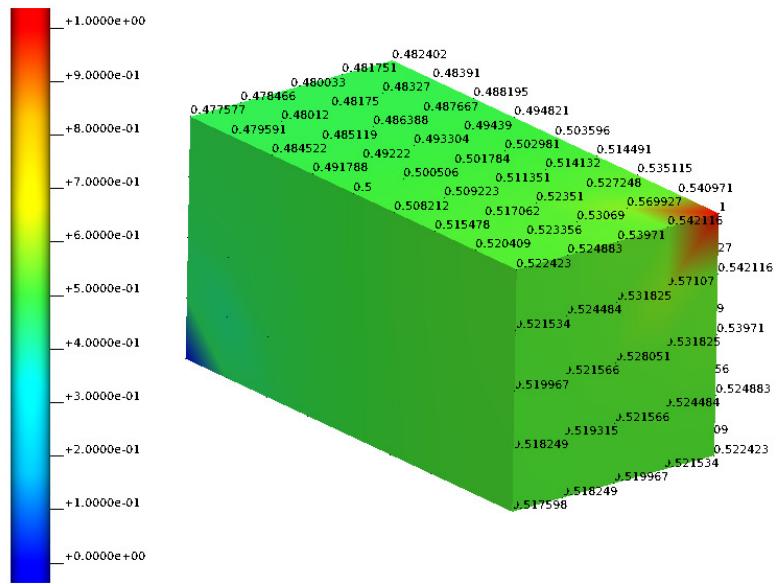


Figure 11: 3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].

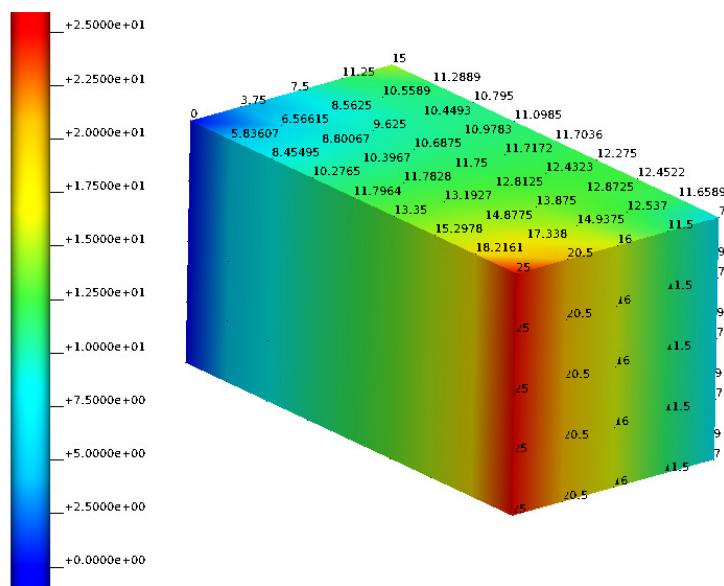


Figure 12: 3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].

4.5 Example-0003 [COMPILES]

Example uses generated regular meshes and solves a static problem, i.e., applies the boundary conditions in one step.

Issues: Not clear how to prescribe Neumann BC. Results seem weird. Reference results are set up.

4.5.1 Mathematical model - 2D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1], \quad (20)$$

with boundary conditions

$$u = 15y \quad x = 0, \quad (21)$$

$$\partial_n u = 25 - 18y \quad x = 2. \quad (22)$$

No material parameters to specify.

4.5.2 Mathematical model - 3D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1] \times [0, 1], \quad (23)$$

with boundary conditions

$$u = 15y \quad x = 0, \quad (24)$$

$$\partial_n u = 25 - 18y \quad x = 2. \quad (25)$$

No material parameters to specify.

4.5.3 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

- Commandline arguments for tests are:

2.0 1.0 0.0 2 1 0 1 0

2.0 1.0 0.0 4 2 0 1 0

2.0 1.0 0.0 8 4 0 1 0

2.0 1.0 0.0 2 1 0 2 0

2.0 1.0 0.0 4 2 0 2 0

```

2.0 1.0 0.0 8 4 0 2 0
2.0 1.0 0.0 2 1 0 1 1
2.0 1.0 0.0 4 2 0 1 1
2.0 1.0 0.0 8 4 0 1 1
2.0 1.0 0.0 2 1 0 2 1
2.0 1.0 0.0 4 2 0 2 1
2.0 1.0 0.0 8 4 0 2 1
2.0 1.0 1.0 2 1 1 1 0
2.0 1.0 1.0 4 2 2 1 0
2.0 1.0 1.0 8 4 4 1 0
2.0 1.0 1.0 2 1 1 2 0
2.0 1.0 1.0 4 2 2 2 0
2.0 1.0 1.0 8 4 4 2 0
2.0 1.0 1.0 2 1 1 1 1
2.0 1.0 1.0 4 2 2 1 1
2.0 1.0 1.0 8 4 4 1 1
2.0 1.0 1.0 2 1 1 2 1
2.0 1.0 1.0 4 2 2 2 1
2.0 1.0 1.0 8 4 4 2 1

```

4.5.4 Result summary

We use CHeart rev. 6292 to produce numerical reference solutions.

Passed tests: 0 / 24

Failed tests:

```

current_run/l2x1x0_n2x1x0_i1_s0/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i1_s0/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i1_s0/Example.part0.exnode
current_run/l2x1x0_n2x1x0_i2_s0/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i2_s0/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i2_s0/Example.part0.exnode
current_run/l2x1x0_n2x1x0_i1_s1/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i1_s1/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i1_s1/Example.part0.exnode
current_run/l2x1x0_n2x1x0_i2_s1/Example.part0.exnode
current_run/l2x1x0_n4x2x0_i2_s1/Example.part0.exnode
current_run/l2x1x0_n8x4x0_i2_s1/Example.part0.exnode
current_run/l2x1x1_n2x1x1_i1_s0/Example.part0.exnode
current_run/l2x1x1_n4x2x2_i1_s0/Example.part0.exnode
current_run/l2x1x1_n8x4x4_i1_s0/Example.part0.exnode
current_run/l2x1x1_n2x1x1_i2_s0/Example.part0.exnode
current_run/l2x1x1_n4x2x2_i2_s0/Example.part0.exnode
current_run/l2x1x1_n8x4x4_i2_s0/Example.part0.exnode
current_run/l2x1x1_n2x1x1_i1_s1/Example.part0.exnode
current_run/l2x1x1_n4x2x2_i1_s1/Example.part0.exnode
current_run/l2x1x1_n8x4x4_i1_s1/Example.part0.exnode
current_run/l2x1x1_n2x1x1_i2_s1/Example.part0.exnode

```

CHeart	- Iron	_2 = 44.2627
CHeart	- Iron	_2 = 37.2770
CHeart	- Iron	_2 = 32.2165
CHeart	- Iron	_2 = 27.3358
CHeart	- Iron	_2 = 22.1869
CHeart	- Iron	_2 = 19.7449
CHeart	- Iron	_2 = 44.2627
CHeart	- Iron	_2 = 37.2770
CHeart	- Iron	_2 = 32.2165
CHeart	- Iron	_2 = 27.3358
CHeart	- Iron	_2 = 22.1869
CHeart	- Iron	_2 = 19.7449
CHeart	- Iron	_2 = 124.749
CHeart	- Iron	_2 = 128.672
CHeart	- Iron	_2 = 143.606
CHeart	- Iron	_2 = 94.2619
CHeart	- Iron	_2 = 98.7606
CHeart	- Iron	_2 = 118.047
CHeart	- Iron	_2 = 124.749
CHeart	- Iron	_2 = 128.672
CHeart	- Iron	_2 = 143.606
CHeart	- Iron	_2 = 94.2619

```
current_run/l2x1x1_n4x2x2_i2_s1/Example.part0.exnode  
current_run/l2x1x1_n8x4x4_i2_s1/Example.part0.exnode
```

| CHeart - Iron |₂ = 98.7606
| CHeart - Iron |₂ = 118.047

Figure 13: 2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].

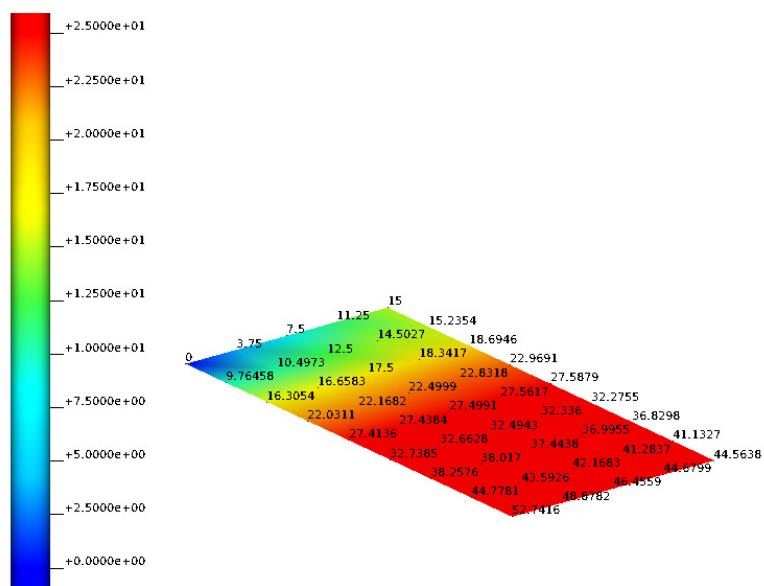


Figure 14: 2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0].

Figure 15: 3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].

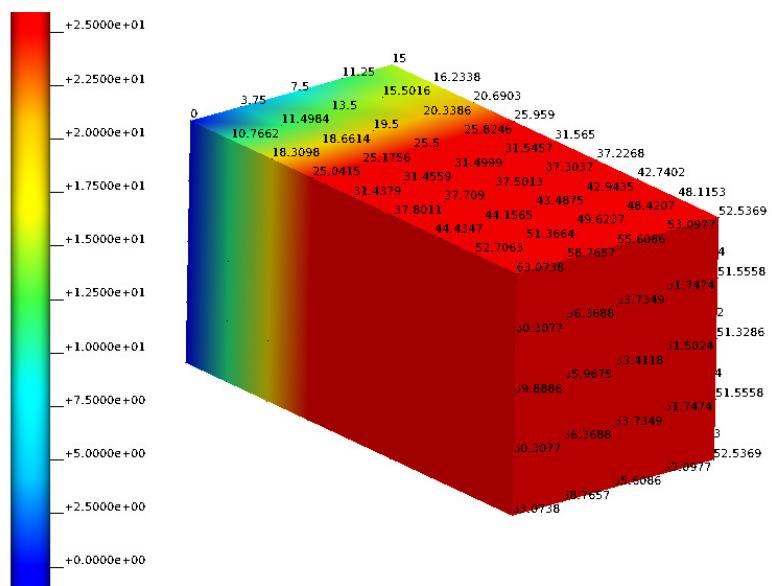


Figure 16: 3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0].

4.6 Example-0004 [VALIDATED]

Example uses generated regular meshes and solves a static problem, i.e., applies the boundary conditions in one step.

4.6.1 Mathematical model - 2D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 2] \times [0, 1], \quad (26)$$

with boundary conditions

$$u = 2.0e^x \cdot \cos(y) \quad \text{on } \partial\Omega. \quad (27)$$

No material parameters to specify.

4.6.2 Computational model

- Commandline arguments are:

integer: number of elements in x-direction
 integer: number of elements in y-direction
 integer: number of elements in z-direction (set to zero for 2D)
 integer: interpolation order (1: linear; 2: quadratic)
 integer: solver type (0: direct; 1: iterative)

- Commandline arguments for tests are:

```
4 2 0 1 0
8 4 0 1 0
2 1 0 2 0
4 2 0 2 0
8 4 0 2 0
4 2 0 1 1
8 4 0 1 1
2 1 0 2 1
4 2 0 2 1
8 4 0 2 1
100 50 0 1 0 (not tested yet..)
100 50 0 2 0 (not tested yet..)
100 50 0 1 1 (not tested yet..)
100 50 0 2 1 (not tested yet..)
```

4.6.3 Result summary

We use CHeart rev. 6292 to produce numerical reference solutions.

Passed tests: 10 / 10

No failed tests.

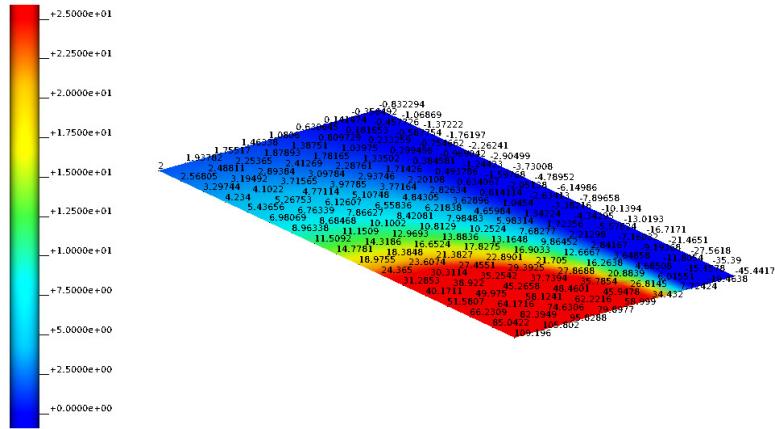


Figure 17: 2D results, iron reference w/ command line arguments [8 4 0 2 0].

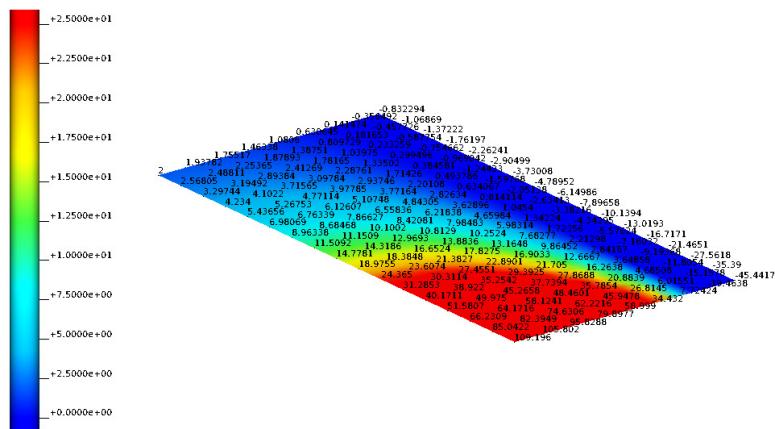


Figure 18: 2D results, current run w/ command line arguments [8 4 0 2 0].

4.7 Example-0005 [VALIDATED]

Example uses two user-defined (2d and 3d) unregular meshes and solves a patch test in form of a static problem, i.e., applies the boundary conditions in one step. Here, a Laplace problem with two Dirichlet boundary conditions is solved. The analytical solution for this setting is known and results in a linear distribution of the scalar Laplace parameter u . The numerical solution has to recover this the linear distribution in order to pass the Patch test. 2d and 3d meshes are according to MacNeil and Harder (1985).

4.7.1 Mathematical model - 2D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 0.24] \times [0, 0.12], \quad (28)$$

with boundary conditions

$$u = 0 \quad x = 0, \quad (29)$$

$$u = 1 \quad x = 0.24. \quad (30)$$

No material parameters to specify.

4.7.2 Mathematical model - 3D

We solve the following scalar equation,

$$\nabla \cdot \nabla u = 0 \quad \Omega = [0, 1] \times [0, 1] \times [0, 1], \quad (31)$$

with boundary conditions

$$u = 0 \quad x = 0, \quad (32)$$

$$u = 1 \quad x = 1. \quad (33)$$

No material parameters to specify.

4.7.3 Computational model

- Commandline arguments are:

integer: dimension (2: 2d; 3: 3d)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

- Commandline arguments for tests are:

2 1 0

2 2 0

2 1 1

2 2 1

3 1 0

3 2 0

3 1 1

3 2 1

4.7.4 Result summary

Since the analytical result is known and the numerical results have to recover the linear distribution of u , the comparisons are done with the analytical solution.

Passed tests: 8 / 8

No failed tests.

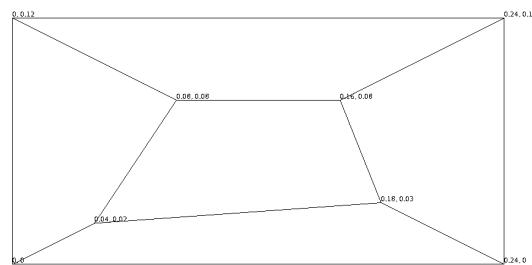


Figure 19: 2D geometry and mesh.

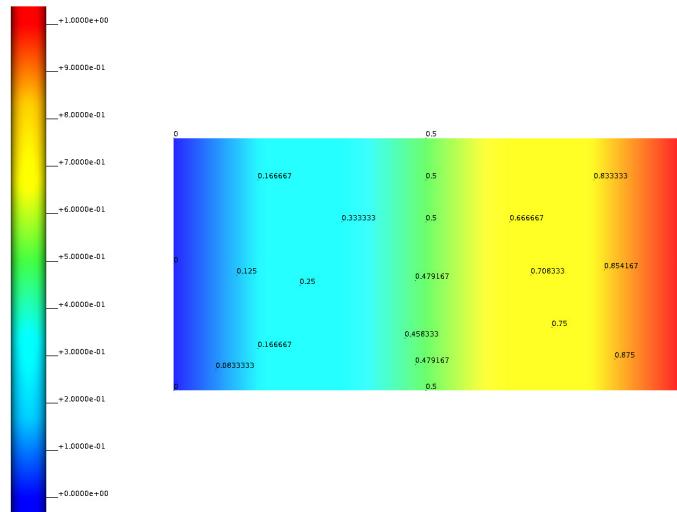


Figure 20: 2D results, iron reference w/ command line arguments [2 2 0].

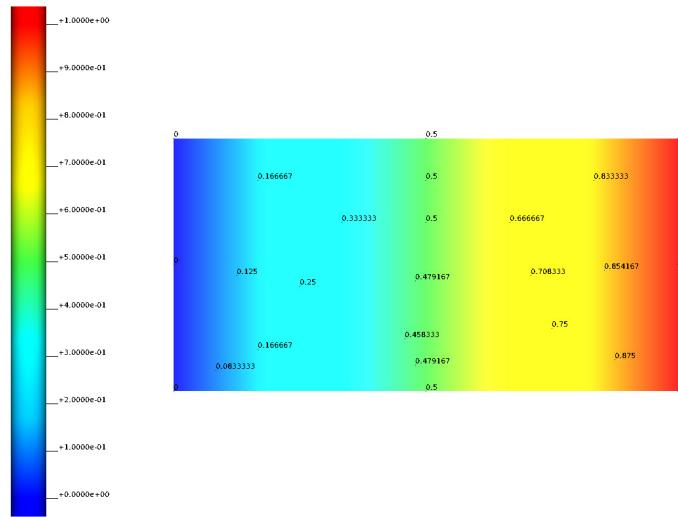


Figure 21: 2D results, current run w/ command line arguments [2 2 0].

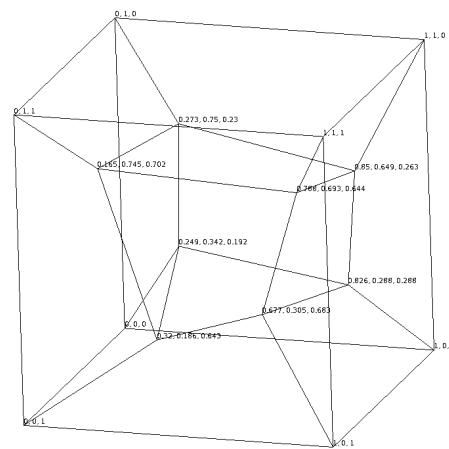


Figure 22: 3D geometry (unit cube) and mesh.

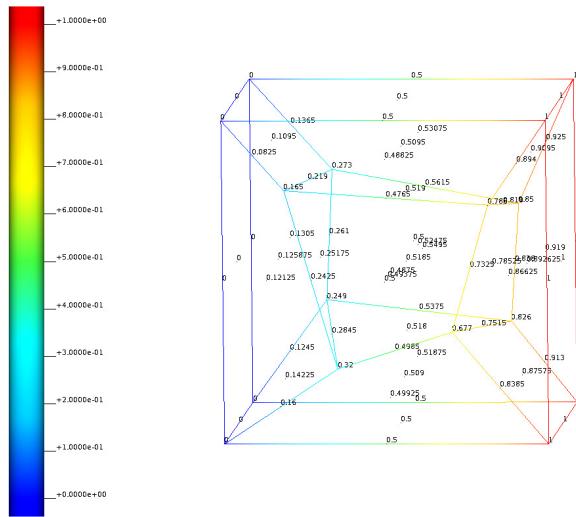


Figure 23: 3D results, iron reference w/ command line arguments [3 2 0].

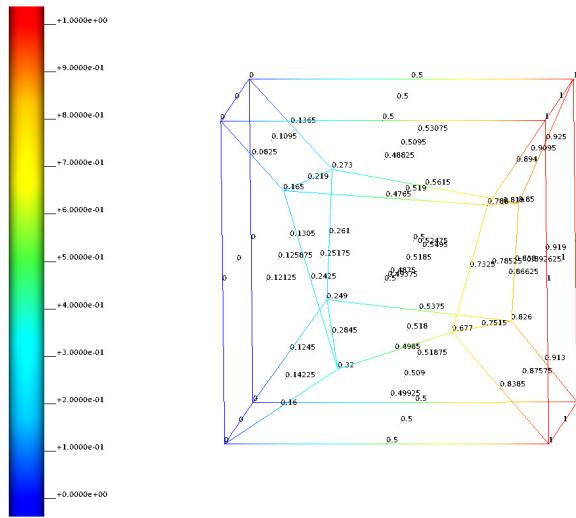


Figure 24: 3D results, current run w/ command line arguments [3 2 0].

4.8 Example-0011 [VALIDATED]

Example uses generated regular meshes and solves a static problem, i.e., applies the boundary conditions in one step.

4.8.1 Mathematical model - 2D

We solve the following scalar equation,

$$\nabla \cdot [\sigma \nabla u] = 0 \quad \Omega = [0, 2] \times [0, 1], \quad (34)$$

with boundary conditions

$$u = 0 \quad x = y = 0, \quad (35)$$

$$u = 1 \quad x = 2, y = 1. \quad (36)$$

The conductivity tensor is defined as,

$$\sigma(x, t) = \sigma = I. \quad (37)$$

4.8.2 Mathematical model - 3D

We solve the following scalar equation,

$$\nabla \cdot [\sigma \nabla u] = 0 \quad \Omega = [0, 2] \times [0, 1] \times [0, 1], \quad (38)$$

with boundary conditions

$$u = 0 \quad x = y = z = 0, \quad (39)$$

$$u = 1 \quad x = 2, y = z = 1. \quad (40)$$

The conductivity tensor is defined as,

$$\sigma(x, t) = \sigma = I. \quad (41)$$

4.8.3 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

float: σ_{11}

float: σ_{22}

float: σ_{33} (ignored for 2D)

- Commandline arguments for tests are:

```

2.0 1.0 0.0 2 1 0 1 0 1 1
2.0 1.0 0.0 4 2 0 1 0 1 1
2.0 1.0 0.0 8 4 0 1 0 1 1
2.0 1.0 0.0 2 1 0 2 0 1 1
2.0 1.0 0.0 4 2 0 2 0 1 1
2.0 1.0 0.0 8 4 0 2 0 1 1
2.0 1.0 0.0 2 1 0 1 1 1 1
2.0 1.0 0.0 4 2 0 1 1 1 1
2.0 1.0 0.0 8 4 0 1 1 1 1
2.0 1.0 0.0 2 1 0 2 1 1 1
2.0 1.0 0.0 4 2 0 2 1 1 1
2.0 1.0 0.0 8 4 0 2 1 1 1
2.0 1.0 1.0 2 1 1 1 0 1 1 1
2.0 1.0 1.0 4 2 2 1 0 1 1 1
2.0 1.0 1.0 8 4 4 1 0 1 1 1
2.0 1.0 1.0 2 1 1 2 0 1 1 1
2.0 1.0 1.0 4 2 2 2 0 1 1 1
2.0 1.0 1.0 8 4 4 2 0 1 1 1
2.0 1.0 1.0 2 1 1 1 1 1 1 1
2.0 1.0 1.0 4 2 2 1 1 1 1 1
2.0 1.0 1.0 8 4 4 1 1 1 1 1
2.0 1.0 1.0 2 1 1 2 1 1 1 1
2.0 1.0 1.0 4 2 2 2 1 1 1 1
2.0 1.0 1.0 8 4 4 2 1 1 1 1

```

4.8.4 *Result summary*

We use CHeart rev. 6292 to produce numerical reference solutions.

Passed tests: 24 / 24

No failed tests.

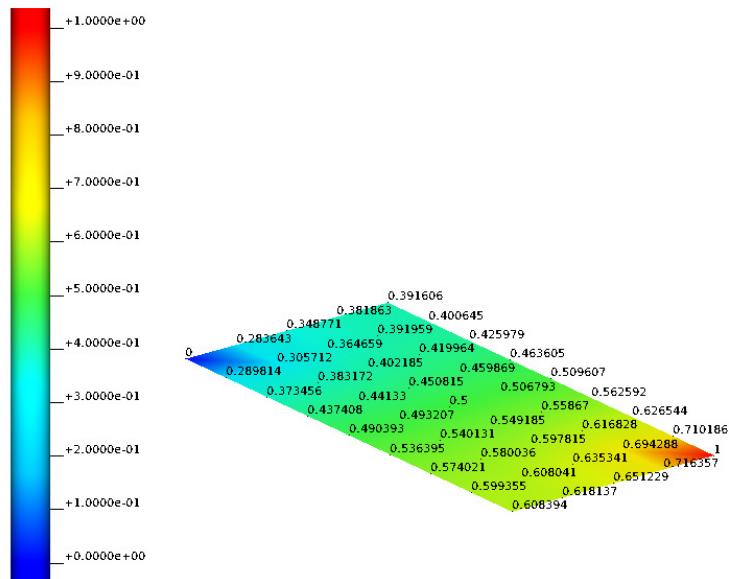


Figure 25: 2D results, iron reference w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0 1].

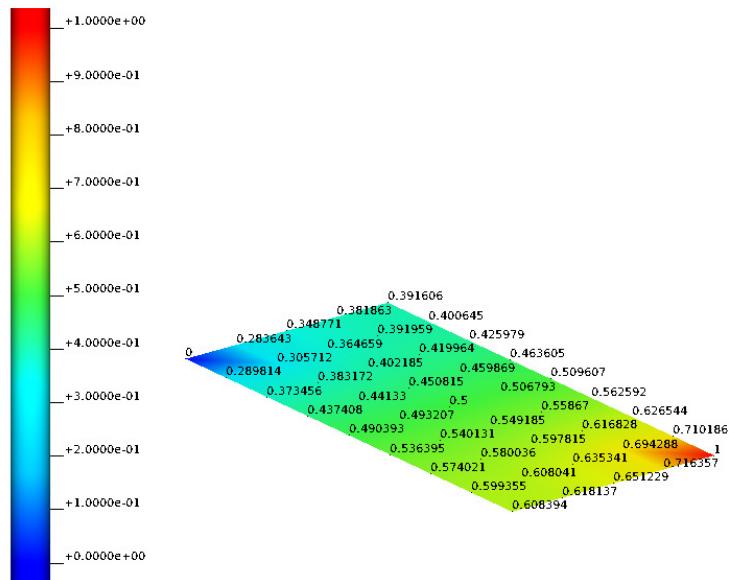


Figure 26: 2D results, current run w/ command line arguments [2.0 1.0 0.0 8 4 0 1 0 1 1].

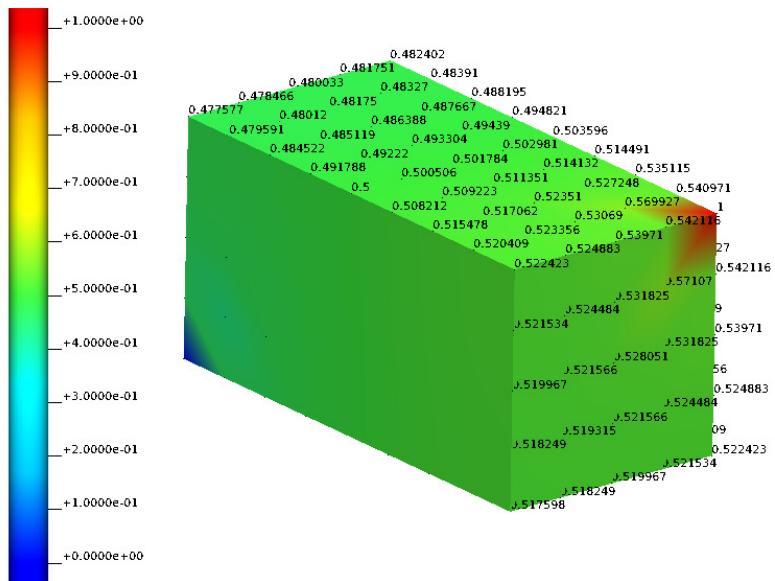


Figure 27: 3D results, iron reference w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0 1 1 1].

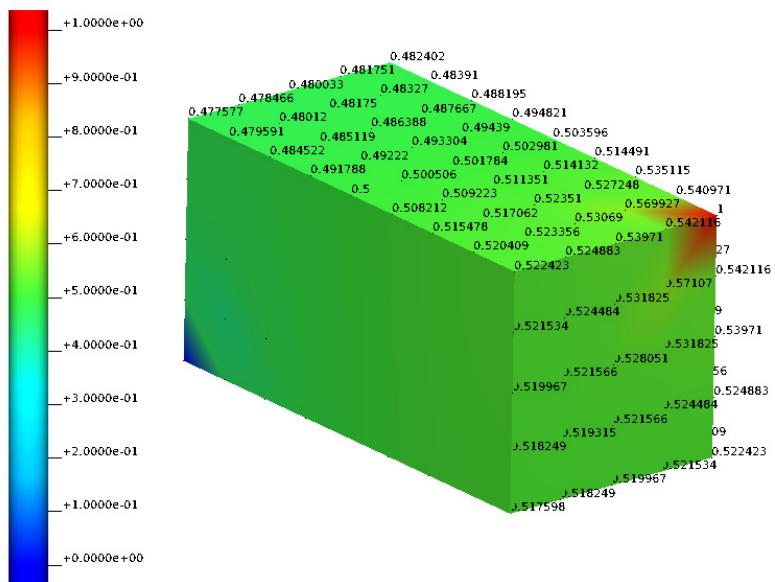


Figure 28: 3D results, current run w/ command line arguments [2.0 1.0 1.0 8 4 4 1 0 1 1 1].

5 LINEAR ELASTICITY

5.1 Equation in general form

$$\partial_{tt} \mathbf{u} + \nabla \cdot \boldsymbol{\sigma}(\mathbf{u}, t) = \mathbf{f}(\mathbf{u}, t) \quad (42)$$

5.2 Example-0101 [PLAUSIBLE]

5.2.1 Mathematical model

We solve the following equation (both 2D and 3D domains are considered),

$$\nabla \cdot \sigma(\mathbf{u}, t) = 0 \quad \Omega = [0, 160] \times [0, 120] \times [0, 120], t \in [0, 5], \quad (43)$$

with time step size $\Delta t = 1$ and $\mathbf{u} = [u_x, u_y]$ in 2D $\mathbf{u} = [u_x, u_y, u_z]$ in 3D. The boundary conditions in 2D are given by

$$u_x = u_y = 0 \quad x = y = 0, \quad (44)$$

$$u_x = 16 \quad x = 160, \quad (45)$$

and in 3D by

$$u_x = u_y = u_z = 0 \quad x = y = z = 0, \quad (46)$$

$$u_x = 16 \quad x = 160. \quad (47)$$

The material parameters are

$$E = 10000 \text{ MPa}, \quad (48)$$

$$\nu = 0.3, \quad (49)$$

$$\rho = 5 \times 10^{-9} \text{ tonne.mm}^3. \quad (50)$$

5.2.2 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

float: elastic modulus

float: Poisson ratio

float: displacement percentage load

- Command line arguments for tests are:

160 120 0 8 6 0 1 0 10000 0.3 0.05

160 120 0 16 12 0 1 0 10000 0.3 0.05

160 120 0 32 24 0 1 0 10000 0.3 0.05

160 120 120 8 6 6 1 0 10000 0.3 0.05

160 120 120 16 12 12 1 0 10000 0.3 0.05

160 120 120 32 24 24 1 0 10000 0.3 0.05

160 120 0 8 6 0 2 0 10000 0.3 0.05

160 120 0 16 12 0 2 0 10000 0.3 0.05

```
160 120 0 32 24 0 2 0 10000 0.3 0.05
160 120 120 8 6 6 2 0 10000 0.3 0.05
160 120 120 16 12 12 2 0 10000 0.3 0.05
160 120 120 32 24 24 2 0 10000 0.3 0.05
```

5.2.3 Results

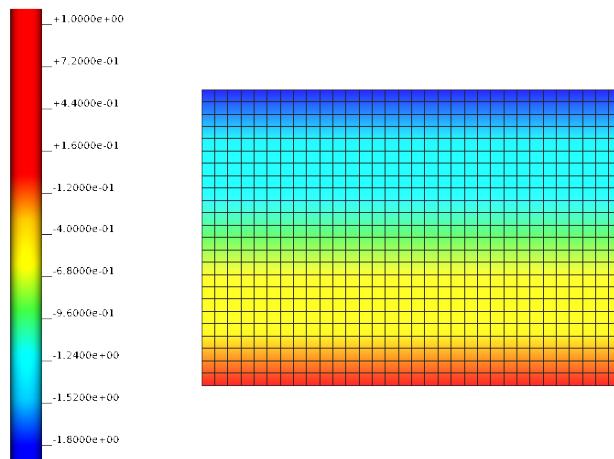


Figure 29: Results, iron 2D fine mesh.

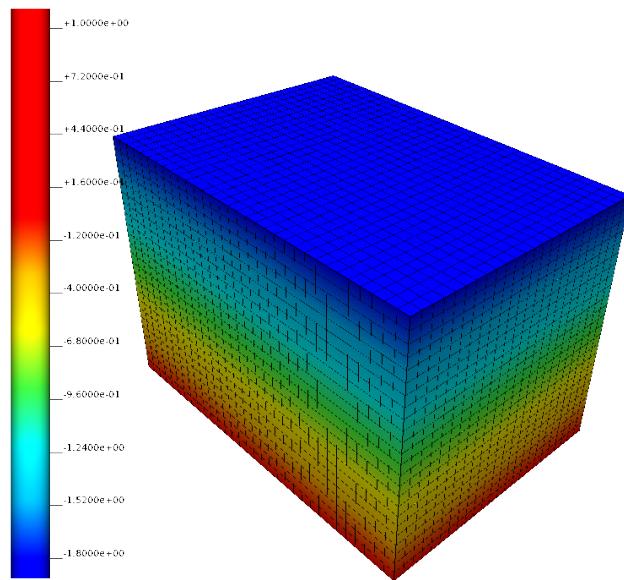


Figure 30: Results, iron 3D fine mesh.

5.2.4 Validation

The iron results are compared to those from Abaqus (version 2017). The figures below show selected results from the validation simulations carried out in Abaqus and provide a qualitative validation. A quantitative validation was carried out by comparing the horizontal displacement u_y along the free-edge ($y = 120$ for 2D and $y = z = 120$ for 3D) and computing the L₂-norm according to

$$L_2\text{-norm} = \frac{1}{N} \times \sum_{i=1}^N \sqrt{(u_{y,\text{abaqus}}^i - u_{y,\text{iron}}^i)^2}, \quad (51)$$

where N is the total number of nodes along the free-edge. The results over the mesh refinements are given in Table 3.

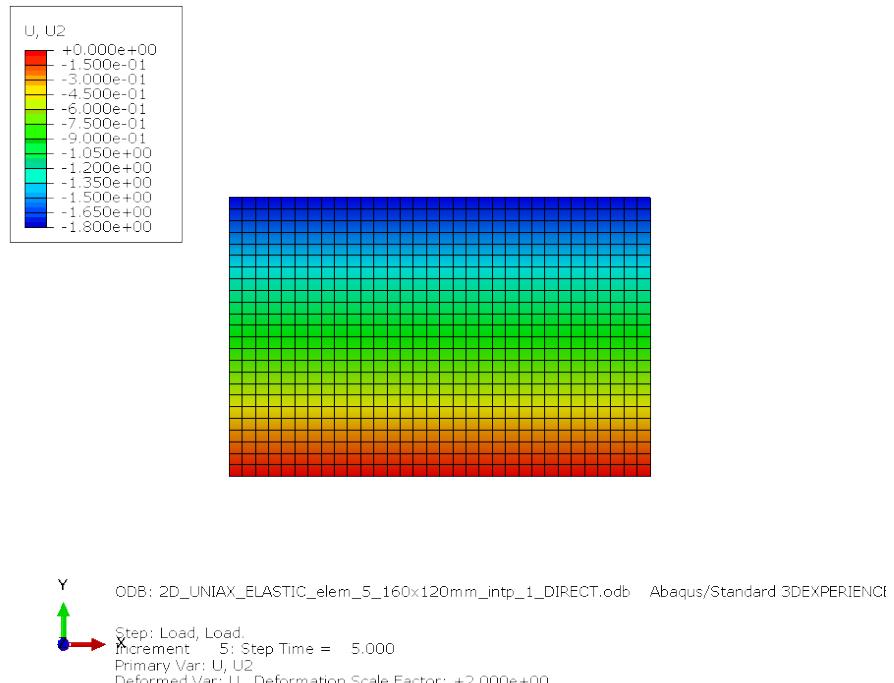


Figure 31: Results, Abaqus 2D fine mesh.

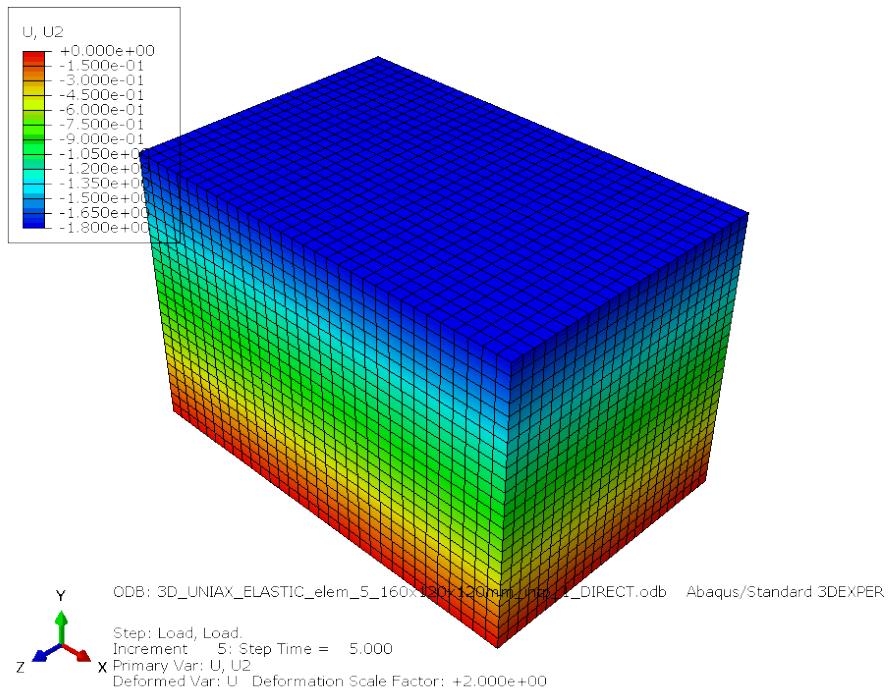


Figure 32: Results, abaqus 3D fine mesh.

Dimension	Mesh	L_2 -norm	Interpolation
2D	Coarse	5.322×10^{-16}	Linear
2D	Medium	1.559×10^{-15}	Linear
2D	Fine	2.900×10^{-15}	Linear
3D	Coarse	3.071×10^{-17}	Linear
3D	Medium	2.125×10^{-17}	Linear
3D	Fine	2.924×10^{-17}	Linear
2D	Coarse	9.728×10^{-16}	Quadratic
2D	Medium	2.039×10^{-15}	Quadratic
2D	Fine	2.159×10^{-15}	Quadratic
3D	Coarse	6.687×10^{-16}	Quadratic
3D	Medium	...	Quadratic
3D	Fine	...	Quadratic

Table 1: Quantitative error between Abaqus 2017 and iron simulations for linear elastic uniaxial extensions

5.3 Example-0102 [PLAUSIBLE]

5.3.1 Mathematical model

We solve the following equation (both 2D and 3D domains are considered),

$$\nabla \cdot \sigma(\mathbf{u}, t) = 0 \quad \Omega = [0, 160] \times [0, 120] \times [0, 120], t \in [0, 5], \quad (52)$$

with time step size $\Delta t = 1$ and $\mathbf{u} = [u_x, u_y]$ in 2D $\mathbf{u} = [u_x, u_y, u_z]$ in 3D. The boundary conditions in 2D are given by

$$u_x = u_y = 0 \quad y = 0, \quad (53)$$

$$u_y = 8 \quad x = 160, \quad (54)$$

and in 3D by

$$u_x = u_z = 0 \quad x = 0, \quad (55)$$

$$u_y = 0 \quad y = 0, \quad (56)$$

$$u_x = 160 \quad x = 160, \quad (57)$$

$$u_y = 8 \quad x = 160. \quad (58)$$

The material parameters are

$$E = 10000 \text{ MPa}, \quad (59)$$

$$\nu = 0.3, \quad (60)$$

$$\rho = 5 \times 10^{-9} \text{ tonne.mm}^3. \quad (61)$$

5.3.2 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

float: elastic modulus

float: Poisson ratio

float: displacement percentage load

- Command line arguments for tests are:

160 120 0 8 6 0 1 0 10000 0.3 0.05

160 120 0 16 12 0 1 0 10000 0.3 0.05

160 120 0 32 24 0 1 0 10000 0.3 0.05

160 120 120 8 6 6 1 0 10000 0.3 0.05

160 120 120 16 12 12 1 0 10000 0.3 0.05

160 120 120 32 24 24 1 0 10000 0.3 0.05

```
160 120 0 8 6 0 2 0 10000 0.3 0.05
160 120 0 16 12 0 2 0 10000 0.3 0.05
160 120 0 32 24 0 2 0 10000 0.3 0.05
160 120 120 8 6 6 2 0 10000 0.3 0.05
160 120 120 16 12 12 2 0 10000 0.3 0.05
160 120 120 32 24 24 2 0 10000 0.3 0.05
```

5.3.3 Results

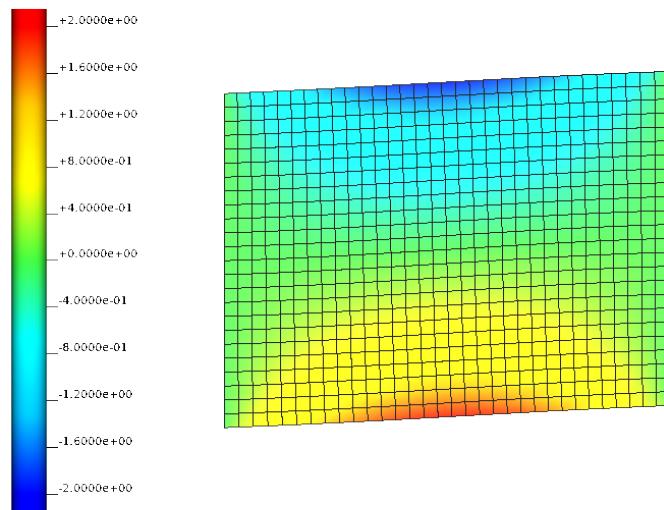


Figure 33: Results, iron 2D fine mesh.

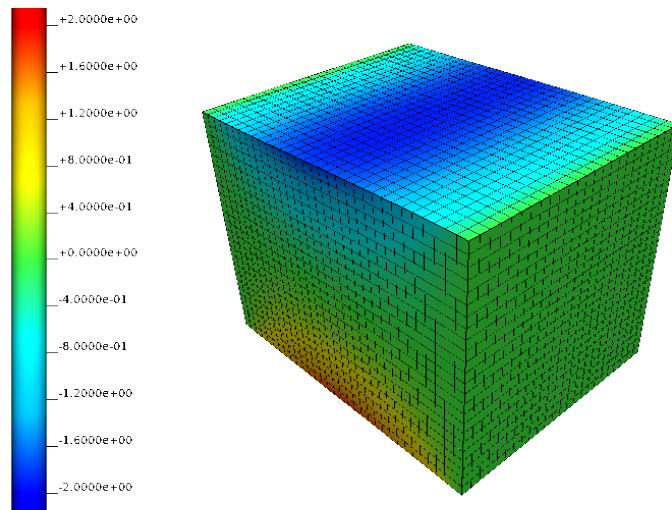


Figure 34: Results, iron 3D fine mesh.

5.3.4 Validation

The iron results are compared to those from Abaqus (version 2017). The figures below show selected results from the validation simulations carried out in Abaqus and provide a qualitative validation. A quantitative validation was carried out by comparing the horizontal displacement u_x along the free-edge ($y = 120$ for 2D and $y = z = 120$ for 3D) and computing the L₂-norm according to

$$L_2\text{-norm} = \frac{1}{N} \times \sum_{i=1}^N \sqrt{(u_{y,\text{abaqus}}^i - u_{y,\text{iron}}^i)^2}, \quad (62)$$

where N is the total number of nodes along the free-edge. The results over the mesh refinements are given in Table 3.

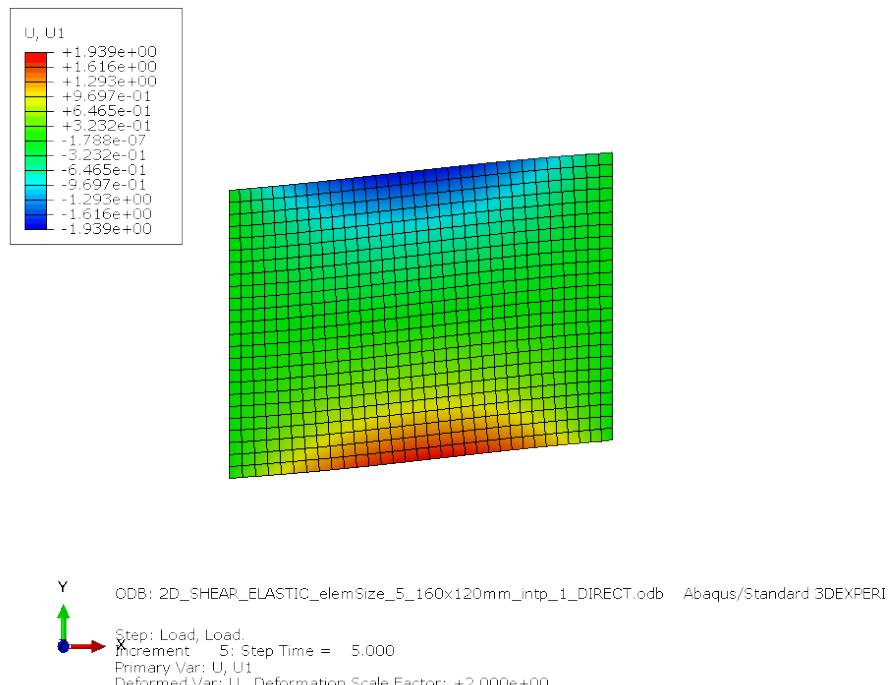


Figure 35: Results, Abaqus 2D fine mesh.

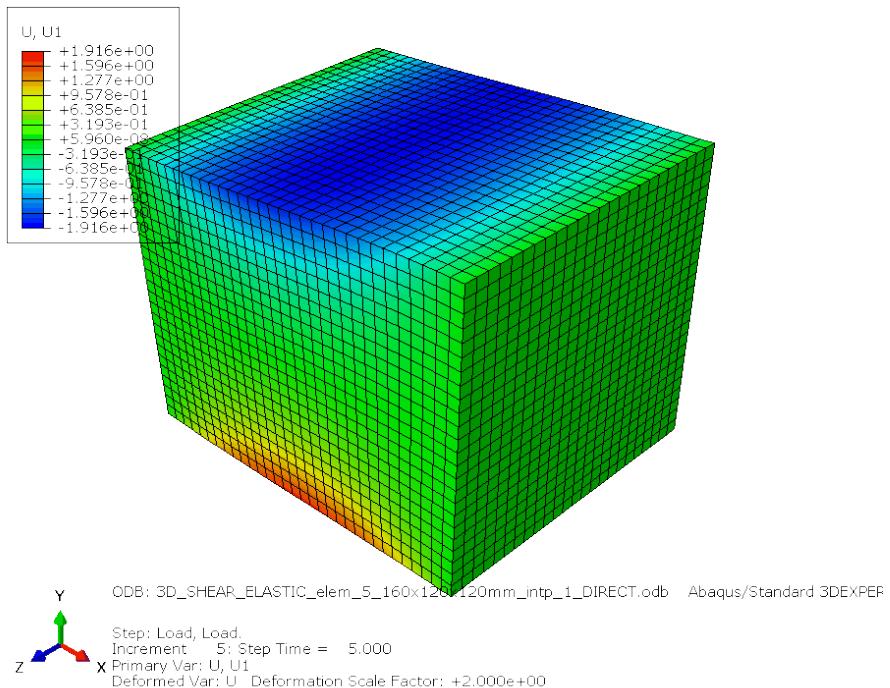


Figure 36: Results, abaqus 3D fine mesh.

Dimension	Mesh	L_2 -norm	Interpolation
2D	Coarse	6.696×10^{-3}	Linear
2D	Medium	1.273×10^{-3}	Linear
2D	Fine	2.489×10^{-4}	Linear
3D	Coarse	4.234×10^{-4}	Linear
3D	Medium	4.184×10^{-5}	Linear
3D	Fine	3.781×10^{-6}	Linear
2D	Coarse	3.036×10^{-4}	Quadratic
2D	Medium	6.099×10^{-5}	Quadratic
2D	Fine	1.089×10^{-5}	Quadratic
3D	Coarse	...	Quadratic
3D	Medium	...	Quadratic
3D	Fine	...	Quadratic

Table 2: Quantitative error between Abaqus 2017 and iron simulations for linear elastic shear

5.4 Example-0111 [PLAUSIBLE]

5.4.1 Mathematical model

We solve the following equation (both 2D and 3D domains are considered),

$$\nabla \cdot \sigma(\mathbf{u}, t) = \mathbf{f}(\mathbf{u}, t) \quad \Omega = [0, 160] \times [0, 120] \times [0, 120], t \in [0, 5], \quad (63)$$

with time step size $\Delta t = 1$ and $\mathbf{u} = [u_x, u_y]$ in 2D $\mathbf{u} = [u_x, u_y, u_z]$ in 3D. The boundary conditions in 2D are given by

$$u_x = u_y = 0 \quad x = y = 0, \quad (64)$$

$$f(u_x) = 6.0 \times 10^4 \quad x = 160, \quad (65)$$

and in 3D by

$$u_x = u_y = u_z = 0 \quad x = y = z = 0, \quad (66)$$

$$f(u_x) = 7.2 \times 10^6 \quad x = 160. \quad (67)$$

The material parameters are

$$E = 10000 \text{ MPa}, \quad (68)$$

$$\nu = 0.3, \quad (69)$$

$$\rho = 5 \times 10^{-9} \text{ tonne.mm}^3. \quad (70)$$

5.4.2 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

float: elastic modulus

float: Poisson ratio

float: XXX

- Command line arguments for tests are:

160 120 0 8 6 0 1 0 10000 0.3 XXX

160 120 0 16 12 0 1 0 10000 0.3 XXX

160 120 0 32 24 0 1 0 10000 0.3 XXX

160 120 120 8 6 6 1 0 10000 0.3 XXX

160 120 120 16 12 12 1 0 10000 0.3 XXX

160 120 120 32 24 24 1 0 10000 0.3 XXX

160 120 0 8 6 0 2 0 10000 0.3 XXX

160 120 0 16 12 0 2 0 10000 0.3 XXX

```
160 120 0 32 24 0 2 0 10000 0.3 XXX
160 120 120 8 6 6 2 0 10000 0.3 XXX
160 120 120 16 12 12 2 0 10000 0.3 XXX
160 120 120 32 24 24 2 0 10000 0.3 XXX
```

5.4.3 Results

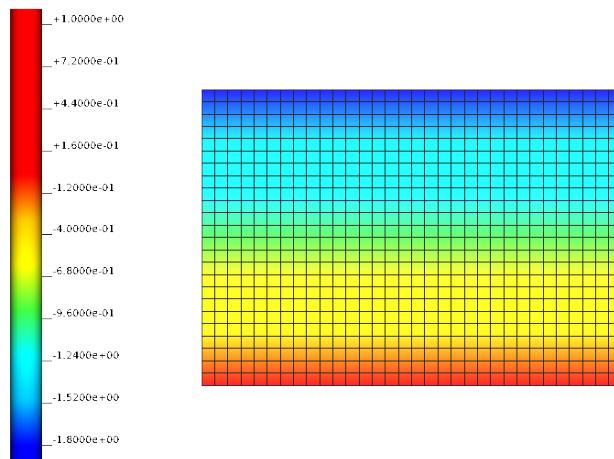


Figure 37: Results, iron 2D fine mesh.

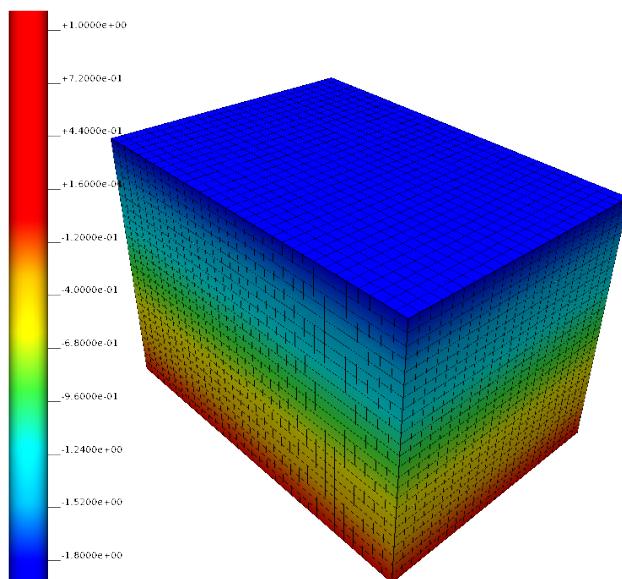


Figure 38: Results, iron 3D fine mesh.

5.4.4 Validation

The iron results are compared to those from Abaqus (version 2017). The figures below show selected results from the validation simulations carried out in Abaqus and provide a qualitative validation. A quantitative validation was carried out by comparing the horizontal displacement u_y along the free-edge ($y = 120$ for 2D and $y = z = 120$ for 3D) and computing the L₂-norm according to

$$L_2\text{-norm} = \frac{1}{N} \times \sum_{i=1}^N \sqrt{(u_{y, \text{abaqus}}^i - u_{y, \text{iron}}^i)^2}, \quad (71)$$

where N is the total number of nodes along the free-edge. The results over the mesh refinements are given in Table 3.

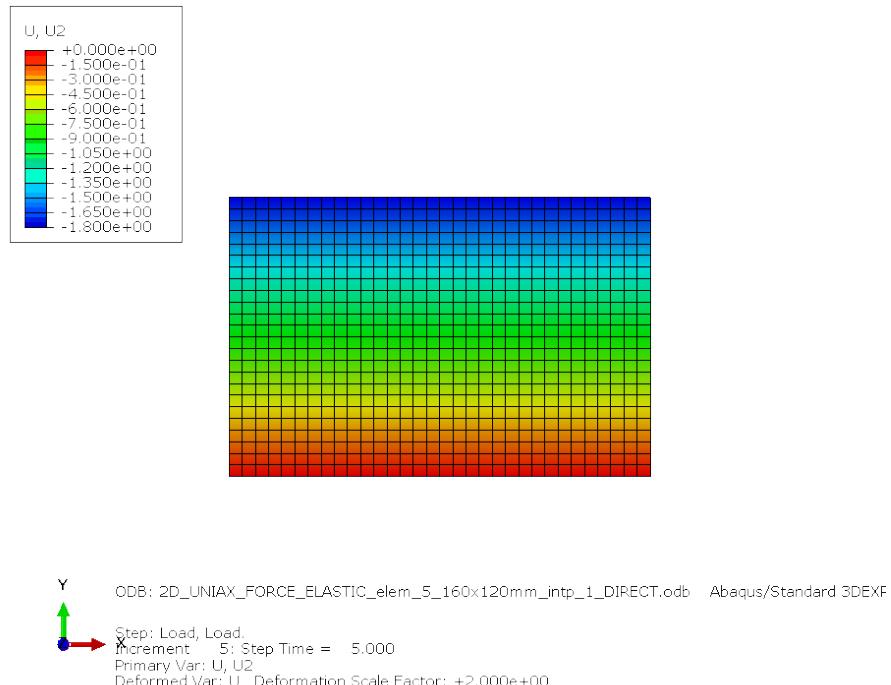


Figure 39: Results, Abaqus 2D fine mesh.

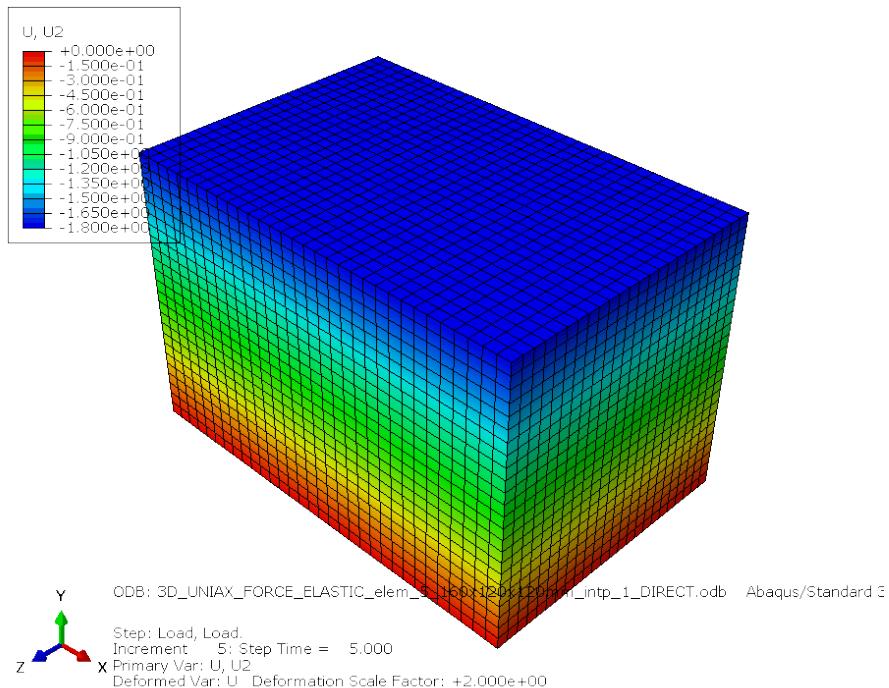


Figure 40: Results, abaqus 3D fine mesh.

Dimension	Mesh	L_2 -norm	Interpolation
2D	Coarse	...	Linear
2D	Medium	...	Linear
2D	Fine	...	Linear
3D	Coarse	...	Linear
3D	Medium	...	Linear
3D	Fine	...	Linear
2D	Coarse	...	Quadratic
2D	Medium	...	Quadratic
2D	Fine	...	Quadratic
3D	Coarse	...	Quadratic
3D	Medium	...	Quadratic
3D	Fine	...	Quadratic

Table 3: Quantitative error between Abaqus 2017 and iron simulations for linear elastic uniaxial extensions

5.5 Example-0112 [PLAUSIBLE]

5.5.1 Mathematical model

We solve the following equation (both 2D and 3D domains are considered),

$$\nabla \cdot \sigma(\mathbf{u}, t) = \mathbf{f}(\mathbf{u}, t) \quad \Omega = [0, 160] \times [0, 120] \times [0, 120], t \in [0, 5], \quad (72)$$

with time step size $\Delta t = 1$ and $\mathbf{u} = [u_x, u_y]$ in 2D $\mathbf{u} = [u_x, u_y, u_z]$ in 3D. The boundary conditions in 2D are given by

$$u_x = u_y = 0 \quad y = 0, \quad (73)$$

$$f(u_y) = 6.0 \times 10^4 \quad x = 160, \quad (74)$$

and in 3D by

$$u_x = u_z = 0 \quad x = 0, \quad (75)$$

$$u_y = 0 \quad y = 0, \quad (76)$$

$$u_x = 160 \quad x = 160, \quad (77)$$

$$f(u_y) = 7.2 \times 10^6 \quad x = 160. \quad (78)$$

The material parameters are

$$E = 10000 \text{ MPa}, \quad (79)$$

$$\nu = 0.3, \quad (80)$$

$$\rho = 5 \times 10^{-9} \text{ tonne.mm}^3. \quad (81)$$

5.5.2 Computational model

- Commandline arguments are:

float: length along x-direction

float: length along y-direction

float: length along z-direction (set to zero for 2D)

integer: number of elements in x-direction

integer: number of elements in y-direction

integer: number of elements in z-direction (set to zero for 2D)

integer: interpolation order (1: linear; 2: quadratic)

integer: solver type (0: direct; 1: iterative)

float: elastic modulus

float: Poisson ratio

float: XXX

- Command line arguments for tests are:

160 120 0 8 6 0 1 0 10000 0.3 XXX

160 120 0 16 12 0 1 0 10000 0.3 XXX

160 120 0 32 24 0 1 0 10000 0.3 XXX

160 120 120 8 6 6 1 0 10000 0.3 XXX

160 120 120 16 12 12 1 0 10000 0.3 XXX

160 120 120 32 24 24 1 0 10000 0.3 XXX

```
160 120 0 8 6 0 2 0 10000 0.3 XXX
160 120 0 16 12 0 2 0 10000 0.3 XXX
160 120 0 32 24 0 2 0 10000 0.3 XXX
160 120 120 8 6 6 2 0 10000 0.3 XXX
160 120 120 16 12 12 2 0 10000 0.3 XXX
160 120 120 32 24 24 2 0 10000 0.3 XXX
```

5.5.3 Results

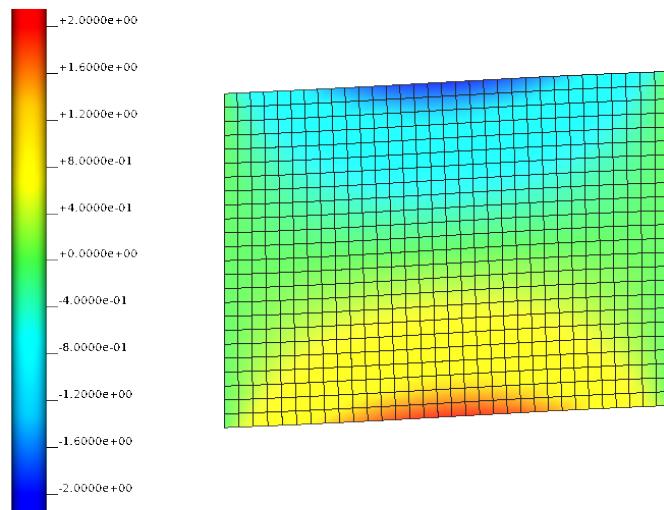


Figure 41: Results, iron 2D fine mesh.

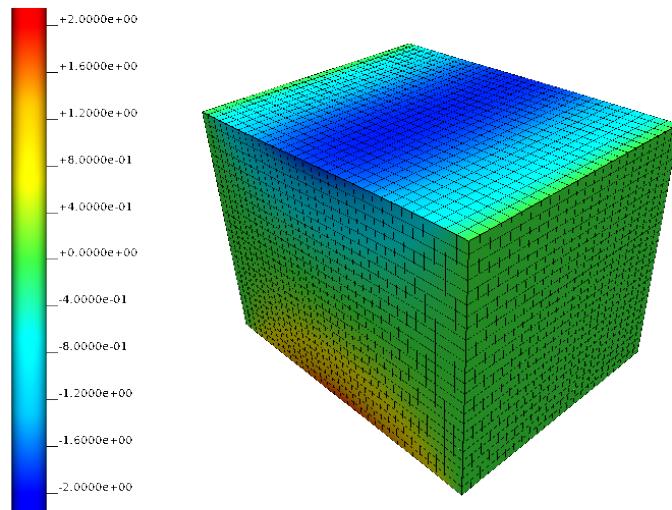


Figure 42: Results, iron 3D fine mesh.

5.5.4 Validation

The iron results are compared to those from Abaqus (version 2017). The figures below show selected results from the validation simulations carried out in Abaqus and provide a qualitative validation. A quantitative validation was carried out by comparing the horizontal displacement u_x along the free-edge ($y = 120$ for 2D and $y = z = 120$ for 3D) and computing the L₂-norm according to

$$L_2\text{-norm} = \frac{1}{N} \times \sum_{i=1}^N \sqrt{(u_{y,\text{abaqus}}^i - u_{y,\text{iron}}^i)^2}, \quad (82)$$

where N is the total number of nodes along the free-edge. The results over the mesh refinements are given in Table 3.

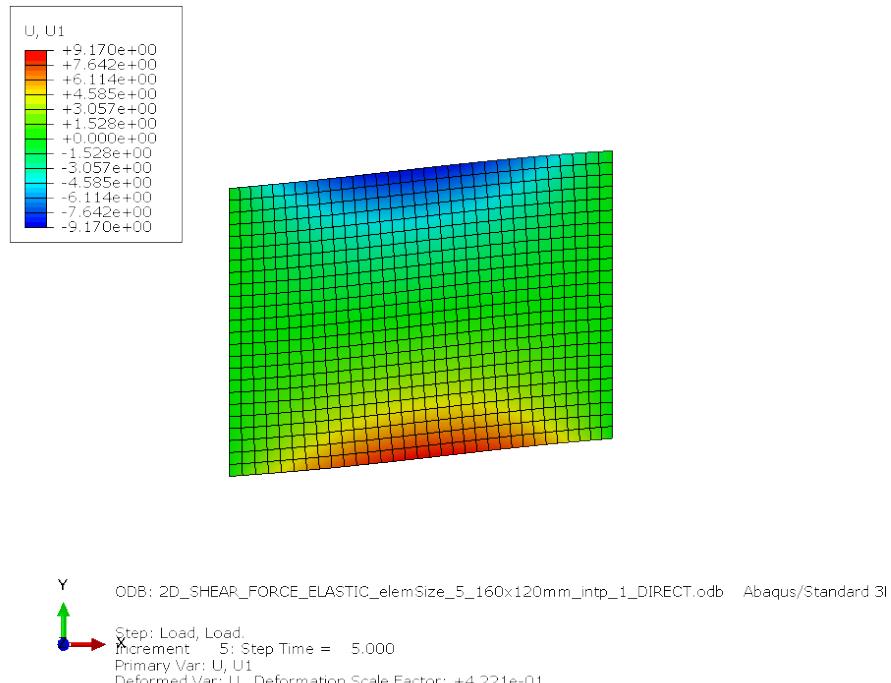


Figure 43: Results, Abaqus 2D fine mesh.

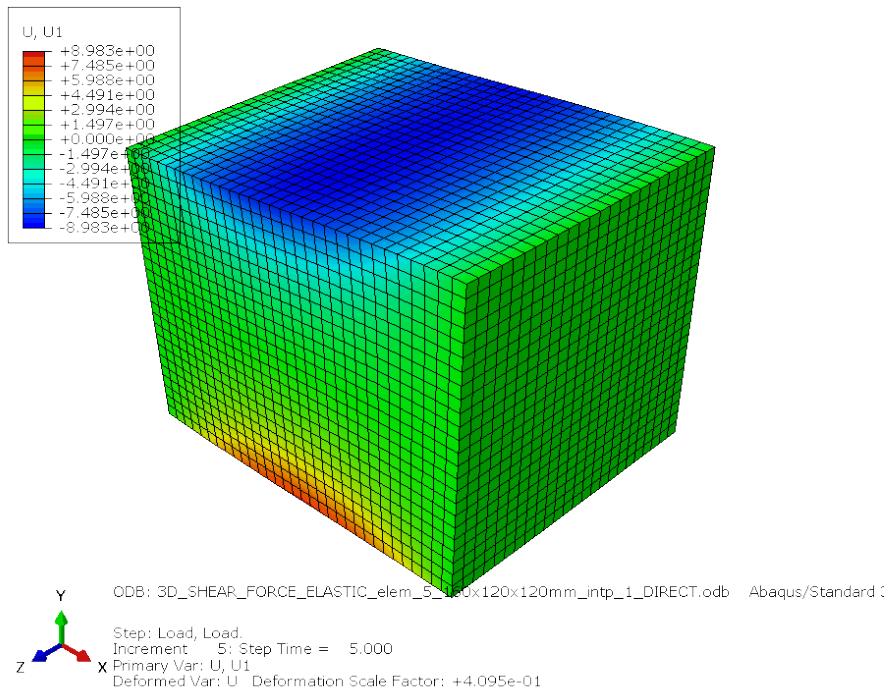


Figure 44: Results, abaqus 3D fine mesh.

Dimension	Mesh	L_2 -norm	Interpolation
2D	Coarse	...	Linear
2D	Medium	...	Linear
2D	Fine	...	Linear
3D	Coarse	...	Linear
3D	Medium	...	Linear
3D	Fine	...	Linear
2D	Coarse	...	Quadratic
2D	Medium	...	Quadratic
2D	Fine	...	Quadratic
3D	Coarse	...	Quadratic
3D	Medium	...	Quadratic
3D	Fine	...	Quadratic

Table 4: Quantitative error between Abaqus 2017 and iron simulations for linear elastic shear

6 FINITE ELASTICITY

7 NAVIER-STOKES FLOW

7.1 Equation in general form

$$\partial_t(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} + p \mathbf{I}) = \rho \mathbf{f} \quad (83)$$

7.2 Example-0302-u [COMPILES]

Example uses user-defined simplex meshes in CHart mesh format with quadratic/linear interpolation for velocity/pressure and solves a dynamic problem.

Setup is the well-known lid-driven cavity problem on the unit square or unit cube in two and three dimensions.

Current issue: does not converge after 30 some time iterations (2D and 3D).

Visualization issue: In exelem-file, replace

1. constant(2)*constant, no modify, grid based.
#xi1=0, #xi2=0
2. constant(2)*constant, no modify, grid based.

with

1. constant*constant, no modify, grid based.
#xi1=0, #xi2=0
2. constant*constant, no modify, grid based.

and likewise for 3D, replace

1. constant(2;3)*constant*constant, no modify, grid based.
#xi1=0, #xi2=0, #xi3=0
2. constant(2;3)*constant*constant, no modify, grid based.

with

1. constant*constant*constant, no modify, grid based.
#xi1=0, #xi2=0, #xi3=0
2. constant*constant*constant, no modify, grid based.

7.2.1 Mathematical model - 2D

We solve the incompressible Navier-Stokes equation,

$$\partial_t(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v}) - \nabla \cdot (\mu \nabla \mathbf{v} - p \mathbf{I}) = \rho \mathbf{f} \quad \Omega = [0, 1] \times [0, 1], \quad (84)$$

$$\nabla \cdot \mathbf{v} = 0, \quad (85)$$

with boundary conditions

$$\mathbf{v} = 0 \quad x = 0, \quad (86)$$

$$\mathbf{v} = 0 \quad x = 1, \quad (87)$$

$$\mathbf{v} = 0 \quad y = 0, \quad (88)$$

$$\mathbf{v} = [1, 0]^T \quad y = 1. \quad (89)$$

Viscosity $\mu = 0.0025$, density $\rho = 1$. Thus, Reynolds number $Re = 400$.

7.2.2 Mathematical model - 3D

We solve the incompressible Navier-Stokes equation,

$$\partial_t(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v}) - \nabla \cdot (\mu \nabla \mathbf{v} - p \mathbf{I}) = \rho \mathbf{f} \quad \Omega = [0, 1] \times [0, 1] \times [0, 1], \quad (90)$$

$$\nabla \cdot \mathbf{v} = 0, \quad (91)$$

with boundary conditions

$$\mathbf{v} = 0 \quad x = 0, \quad (92)$$

$$\mathbf{v} = 0 \quad x = 1, \quad (93)$$

$$\mathbf{v} = 0 \quad y = 0, \quad (94)$$

$$\mathbf{v} = [1, 0]^T \quad y = 1, \quad (95)$$

$$\mathbf{v} = 0 \quad z = 0, \quad (96)$$

$$\mathbf{v} = 0 \quad z = 1. \quad (97)$$

Viscosity $\mu = 0.01$, density $\rho = 1$. Thus, Reynolds number $Re = 100$.

7.2.3 Computational model

- Commandline arguments are:

integer: number of dimensions (2: 2D, 3: 3D)

integer: mesh refinement level (1, 2, 3, ...)

float: start time

float: stop time

float: time step size

float: density

float: viscosity

integer: solver type (0: direct; 1: iterative)

- Commandline arguments for tests are:

2 1 0.0 1.0 0.001 0.0025 1.0 0

2 2 0.0 1.0 0.001 0.0025 1.0 0

2 3 0.0 1.0 0.001 0.0025 1.0 0

2 1 0.0 1.0 0.001 0.0025 1.0 1

2 2 0.0 1.0 0.001 0.0025 1.0 1

2 3 0.0 1.0 0.001 0.0025 1.0 1

3 1 0.0 1.0 0.001 0.01 1.0 0

3 2 0.0 1.0 0.001 0.01 1.0 0

3 3 0.0 1.0 0.001 0.01 1.0 0

3 1 0.0 1.0 0.001 0.01 1.0 1

3 2 0.0 1.0 0.001 0.01 1.0 1

3 3 0.0 1.0 0.001 0.01 1.0 1

- Note: Binary uses command line arguments to search for the relevant mesh files.

7.2.4 *Result summary*

We use CHeart rev. 6292 to produce numerical reference solutions.

Passed tests: 0 / 12

All tests failed.

8 MONODOMAIN

8.1 Example-0401 [PLAUSIBLE]

8.1.1 Mathematical model

We solve the Monodomain Equation

$$\sigma \Delta V_m(t) = A_m \left(C_m \frac{\partial V_m}{\partial t} + I_{\text{ionic}}(V_m) \right) \quad \Omega = [0, 1] \times [0, 1], \quad t \in [0, 3.0] \quad (98)$$

where $V_m(t)$ is given by the Hodgkin-Huxley system of ODEs [2]
with boundary conditions

$$V_m = 0 \quad x = y = 0, \quad (99)$$

$$V_m = 0 \quad x = y = 1. \quad (100)$$

and initial values

$$V_m(t = 0) = -75$$

Additionally a stimulation current I_{stim} is applied for $t_{\text{stim}} = [0, 0.1]$ at the center node of the domain (i.e. at $(x, y) = (\frac{1}{2}, \frac{1}{2},)$).

Material parameters:

$$\sigma = 3.828$$

$$A_m = 500$$

$$C_m = 0.58 \quad \text{for the slow-twitch case}, \quad C_m = 1.0 \quad \text{for the fast-twitch case}$$

$$I_{\text{stim}} = 1200 \quad \text{for the slow-twitch case}, \quad I_{\text{stim}} = 2000.0 \quad \text{for the fast-twitch case}$$

8.1.2 Computational model

- This example uses generated meshes

- Commandline arguments are:

number elements X

number elements Y

interpolation order (1: linear; 2: quadratic)

solver type (0: direct; 1: iterative)

PDE step size

stop time

output frequency

CellML Model URL

slow-twitch

ODE time-step

- Commands for tests are:

```
./folder/src/example 24 24 1 0 0.005 3.0 1 hodgkin_huxley_1952.cellml F 0.0001
./folder/src/example 24 24 1 0 0.005 3.0 1 hodgkin_huxley_1952.cellml F 0.005
./folder/src/example 10 10 1 0 0.005 3.0 1 hodgkin_huxley_1952.cellml F 0.0001
mpirun -n 2 ./folder/src/example 24 24 1 0 0.005 3.0 1 hodgkin_huxley_1952.cellml
```

```
mpirun -n 8 ./folder/src/example 24 24 1 0 0.005 3.0 1 hodgkin_huxley_1952.cellml
./folder/src/example 2 2 1 0 0.005 3.0 1 hodgkin_huxley_1952.cellml F 0.0001
mpirun -n 2 ./folder/src/example 2 2 1 0 0.005 3.0 1 hodgkin_huxley_1952.cellml F
```

- This is a dynamic problem.

8.1.3 *Results*

Passed tests: 36 / 36

No failed tests.

Figure 45: Results movie, 24×24 elements (only works in certain pdf viewers, e.g. Adobe Acrobat Reader)

You get a better understanding of the solutions by looking at the automatically generated animations in `iron-tests/examples/example-0401/doc/figures`.

8.1.4 *Validation*

We compare with a Matlab implementation.

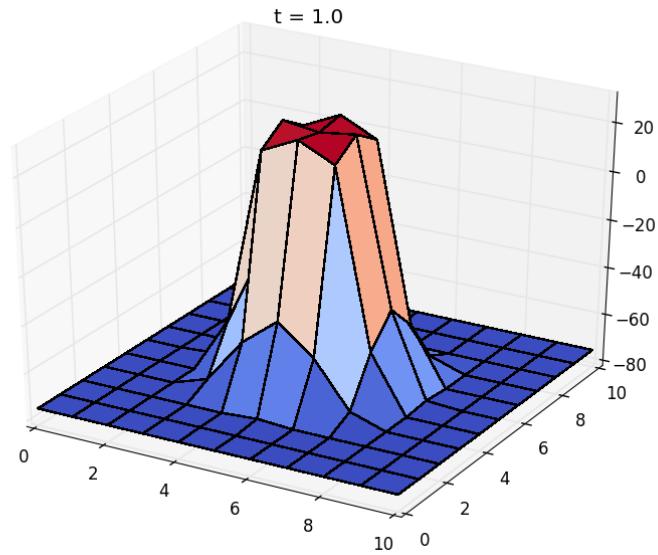


Figure 46: Results, 10×10 elements, $t = 200$

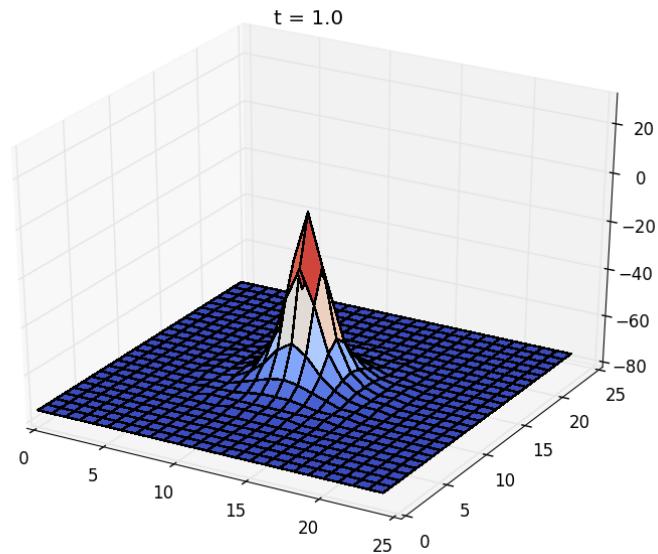


Figure 47: Results, 24×24 elements, $t = 200$

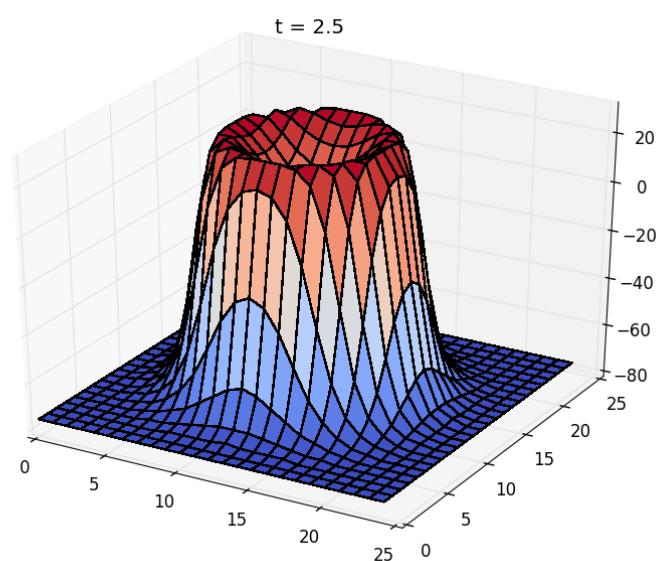


Figure 48: Results, 24×24 elements, $t = 500$

8.2 Example-0402 [PLAUSIBLE]

8.2.1 Mathematical model

We solve the Monodomain Equation

$$\sigma \Delta V_m(t) = A_m \left(C_m \frac{\partial V_m}{\partial t} + I_{ionic}(V_m) \right) \quad \Omega = [0, 1] \times [0, 1], \quad t \in [0, 3.0] \quad (101)$$

where $V_m(t)$ is given by the CellML description of Noble's 1998 improved guinea-pig ventricular cell model system of ODEs [3]

with boundary conditions

$$V_m = 0 \quad x = y = 0, \quad (102)$$

$$V_m = 0 \quad x = y = 1. \quad (103)$$

and initial values

$$V_m(t = 0) = -75$$

Additionally a stimulation current I_{stim} is applied for $t_{stim} = [0, 0.1]$ at the center node of the domain (i.e. at $(x, y) = (\frac{1}{2}, \frac{1}{2})$).

Material parameters:

$$\sigma = 3.828$$

$$A_m = 500$$

$$C_m = 0.58 \quad \text{for the slow-twitch case,} \quad C_m = 1.0 \quad \text{for the fast-twitch case}$$

$$I_{stim} = 1200 \quad \text{for the slow-twitch case,} \quad I_{stim} = 2000.0 \quad \text{for the fast-twitch case}$$

8.2.2 Computational model

- This example uses generated meshes

- Commandline arguments are:

number elements X

number elements Y

interpolation order (1: linear; 2: quadratic)

solver type (0: direct; 1: iterative)

PDE step size

stop time

output frequency

CellML Model URL

slow-twitch

ODE time-step

- Commands for tests are:

`./folder/src/example 24 24 1 0 0.005 3.0 1 n98.xml F 0.0001`

`./folder/src/example 24 24 1 0 0.005 3.0 1 n98.xml F 0.005`

`./folder/src/example 10 10 1 0 0.005 3.0 1 n98.xml F 0.0001`

```

mpirun -n 2 ./folder/src/example 24 24 1 0 0.005 3.0 1 n98.xml F 0.0001
mpirun -n 8 ./folder/src/example 24 24 1 0 0.005 3.0 1 n98.xml F 0.0001
./folder/src/example 2 2 1 0 0.005 3.0 1 n98.xml F 0.0001
mpirun -n 2 ./folder/src/example 2 2 1 0 0.005 3.0 1 n98.xml F 0.0001

```

- This is a dynamic problem.

8.2.3 Results

Passed tests: 36 / 36

No failed tests.

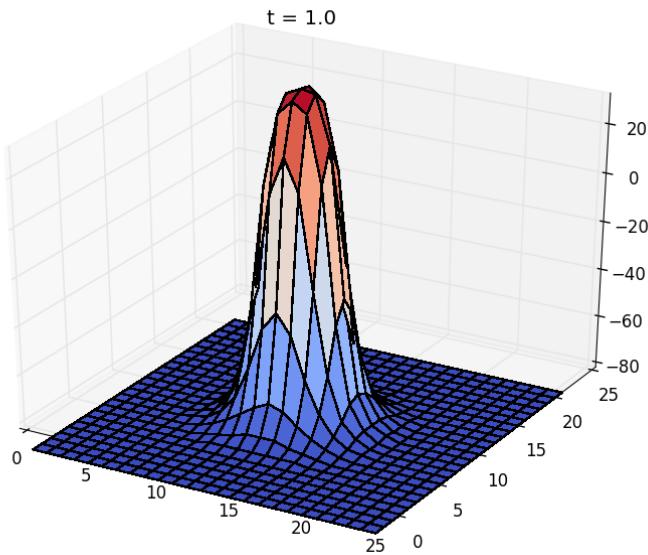


Figure 49: Results, 24×24 elements, $t = 200$

You get a better understanding of the solutions by looking at the automatically generated animations in `iron-tests/examples/example-0402/doc/figures`.

8.2.4 Validation

We compare with a Matlab implementation.

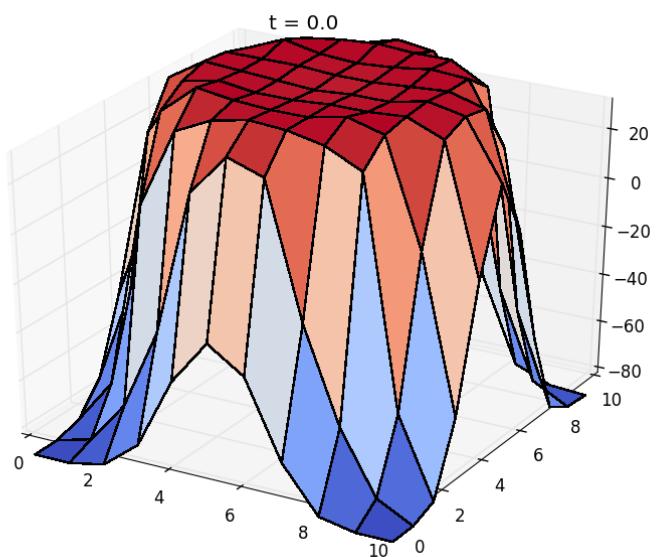


Figure 50: Results, 10×10 elements

8.3 Example-0404-c [PLAUSIBLE]

8.3.1 Mathematical model

We solve the Monodomain Equation

$$\sigma \Delta V_m(t) = A_m \left(C_m \frac{\partial V_m}{\partial t} + I_{ionic}(V_m) \right) \quad \Omega = [0, 1], \quad t \in [0, 10.0] \quad (104)$$

where $V_m(t)$ is given by the Hodgkin-Huxley system of ODEs [2]
with Neumann boundary conditions

$$\frac{\partial u}{\partial n} = 0 \quad x = 0, \quad (105)$$

$$\frac{\partial u}{\partial n} = 0 \quad x = 1. \quad (106)$$

and initial values

$$V_m(t = 0) = -75$$

Additionally a stimulation current I_{stim} is applied for $t_{stim} = [0, 0.5]$ at the center node of the domain (i.e. at $(x, y) = (\frac{1}{2}, \frac{1}{2},)$).

Material parameters:

$$\sigma = 3.828$$

$$A_m = 500$$

$$C_m = 0.58 \quad \text{for the slow-twitch case,} \quad C_m = 1.0 \quad \text{for the fast-twitch case}$$

$$I_{stim} = \begin{cases} 75/10 \cdot (2X) & \text{for } X \geq 10 \text{ reference elements} \\ 75 & \text{for } < 10 \text{ reference elements} \end{cases} \quad \text{for the slow-twitch case,}$$

$$I_{stim} = \begin{cases} 75/12 \cdot (2X) & \text{for } X \geq 12 \text{ reference elements} \\ 75 & \text{for } < 12 \text{ reference elements} \end{cases} \quad \text{for the fast-twitch case}$$

8.3.2 Computational model

- This example uses generated meshes

- Commandline arguments are:

number of elements

order of interpolation

solver type (0: direct; 1: iterative)

time step PDE

end time

output file stride

cellml model file

if slow-twitch (T: slow-twitch, F: fast-twitch)

time step ODE

- Commandline arguments for tests are:

64 2 0 0.01 10 5 hodgkin_huxley_1952.cellml F 0.01

64 2 0 0.005 10 10 hodgkin_huxley_1952.cellml F 0.005

```

64 2 0 0.001 10 50 hodgkin_huxley_1952.cellml F 0.001
64 2 0 0.0005 10 100 hodgkin_huxley_1952.cellml F 0.0005
64 2 0 0.00025 10 200 hodgkin_huxley_1952.cellml F 0.00025

```

- This is a dynamic problem.
- More test cases for 2nd order should be constructed.

8.3.3 Results

We run the scenario for different time step widths and examine the experimental order of convergence.

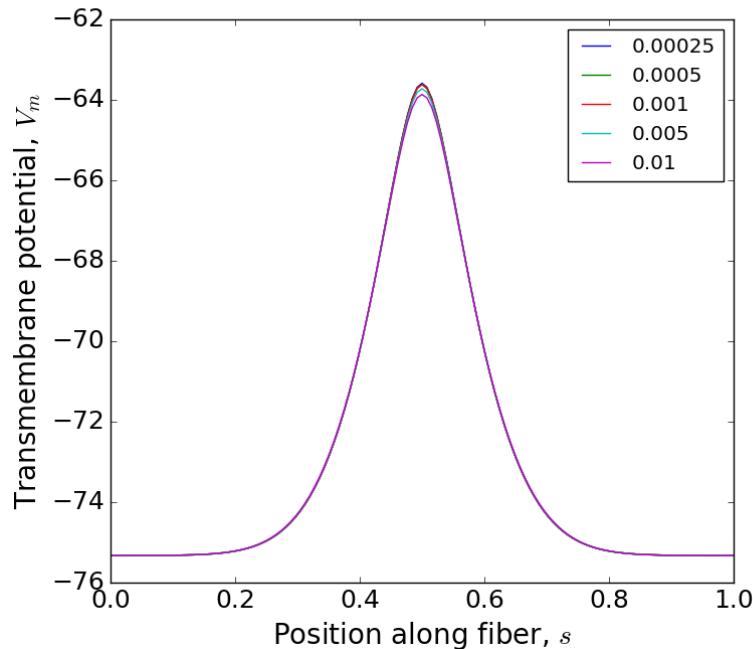


Figure 51: V_m for time $t = 1.0$, different time step widths
 $dt \in \{0.01, 0.005, 0.001, 0.0005, 0.00025\}$

8.3.4 Validation

The purpose of this test case is to see if convergence orders are as expected, no actual validation of the output takes place.

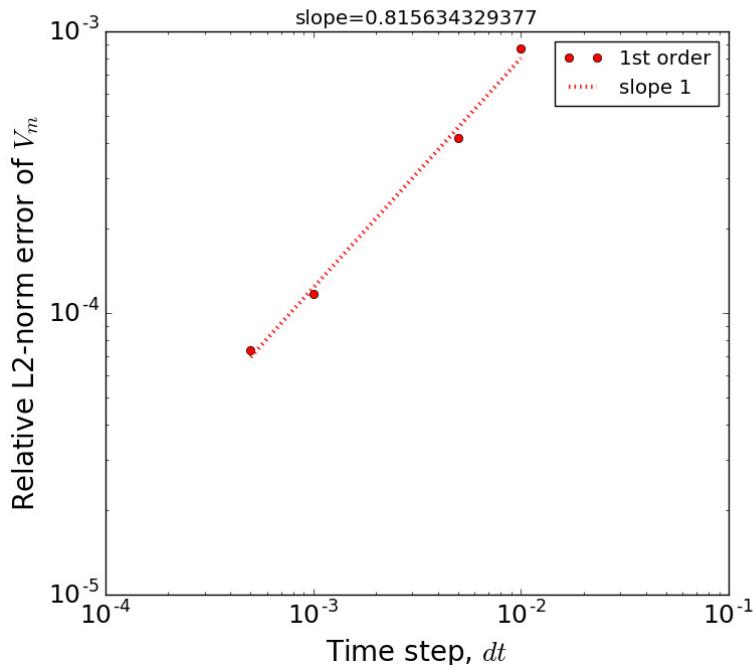


Figure 52: Error at $t = 1.0$ for different time steps widths. The slope (=experimental order of convergence) should be around 1.

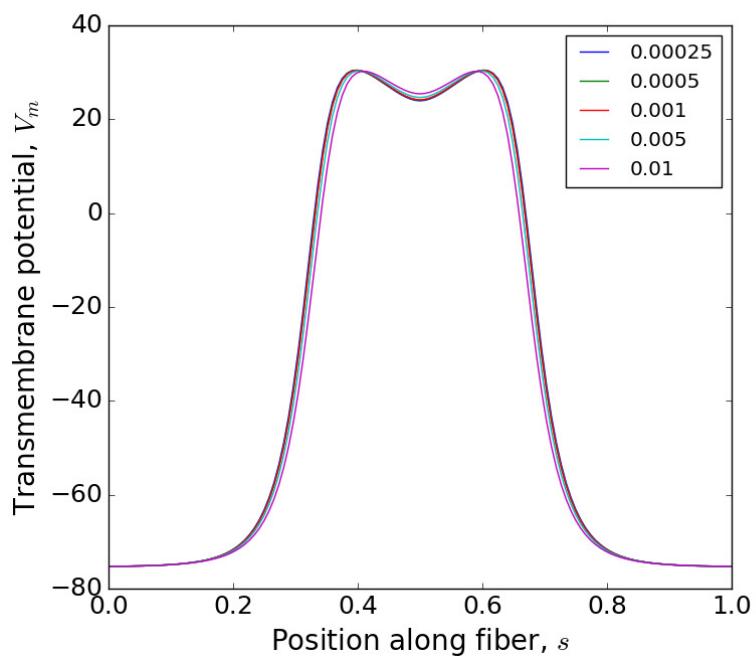


Figure 53: V_m for time $t = 3.0$, different time step widths $dt \in \{0.01, 0.005, 0.001, 0.0005, 0.00025\}$

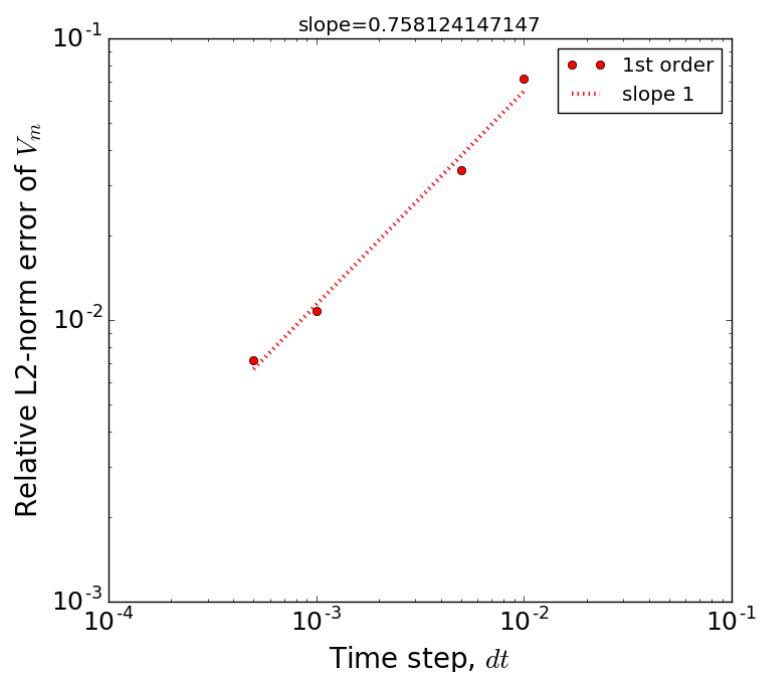


Figure 54: Error at $t = 3.0$ for different time steps widths. The slope (=experimental order of convergence) should be around 1.

9 CELLML MODEL

REFERENCES

- [1] Chris Bradley, Andy Bowery, Randall Britten, Vincent Budelmann, Oscar Camara, Richard Christie, Andrew Cookson, Alejandro F Frangi, Thiranja Babarenda Gamage, Thomas Heidlauf, et al. OpenCMISS: a multi-physics & multi-scale computational infrastructure for the vph-/physiome project. *Progress in biophysics and molecular biology*, 107(1):32–47, 2011.
- [2] Alan L Hodgkin and Andrew F Huxley. Propagation of electrical signals along giant nerve fibres. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, pages 177–183, 1952.
- [3] Denis Noble, Anthony Varghese, Peter Kohl, and Penelope Noble. Improved guinea-pig ventricular cell model incorporating a diadic space, ikr and iks, and length-and tension-dependent processes. *Canadian Journal of Cardiology*, 14(1):123–134, 1998.