|  |  |
| --- | --- |
|  | 4/4/2020  Bound Metal Deposition |
| PHASE-FIELD in MOOSE | |
|  | |
| DGL, VH, KRW  TP project | |

# Contents

[Contents 2](#_Toc41835409)

[List of Figures 3](#_Toc41835410)

[List of Tables 3](#_Toc41835411)

[I. Model development in MOOSE 4](#_Toc41835412)

[A. Adding one more particle in 2D (sint\_t1\_isomob\_3gr.i) 4](#_Toc41835413)

[B. Coupling to elastic stress (sint\_t1\_isom\_3gr\_el\_3S.i) 5](#_Toc41835414)

[C. Increase dimension to 3D (sint\_3gr\_3.i) 6](#_Toc41835415)

[1. Grain growth 3D (grain\_growth\_3D\_random.i). 7](#_Toc41835416)

[2. Changing Conserved.i to 3D 7](#_Toc41835417)

[3. Changing location of the particles in sint\_3gr\_3 8](#_Toc41835418)

[D. 2D model with more than 3 particles 8](#_Toc41835419)

[E. Adding periodic boundary conditions 9](#_Toc41835420)

[F. Gravity (sint1\_isom\_9gr\_el\_2d\_PG\_02\_m3\_0.i) 10](#_Toc41835421)

[G. UNITS 12](#_Toc41835422)

[1. Phase field 12](#_Toc41835423)

[2. A, B, M, L and coefficients 13](#_Toc41835424)

[3. Mechanics 14](#_Toc41835425)

[4. Modifying code 15](#_Toc41835426)

[H. Example. Spinodal decomposition (https://mooseframework.inl.gov/old/wiki/MooseTutorials/IronChromiumDecomposition/) 16](#_Toc41835427)

[1. Units 16](#_Toc41835428)

[2. S1\_testmodel.i (s1\_testmodel.i) 16](#_Toc41835429)

[3. S2\_testmodel.i (s2\_fasttest.i) 17](#_Toc41835430)

[I. A phase field model of pressure-assisted sintering (Dzepina et al., 2019) 17](#_Toc41835431)

[II. Thermomechanical coupling 17](#_Toc41835432)

[A. Simple examples 17](#_Toc41835433)

[1. circle\_thermal\_expansion\_stress.i (combined examples) 17](#_Toc41835434)

[B. Darcy Flow Tutorial 18](#_Toc41835435)

[1. Equations of motion (https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Problem/) 18](#_Toc41835436)

[2. Steps 18](#_Toc41835437)

[a) Step 01 (https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Step01/) 18](#_Toc41835438)

[b) Step 2 (https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Step02/ ) 19](#_Toc41835439)

[c) Step 3 (https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Step03/ ) 19](#_Toc41835440)

[d) Step 4 19](#_Toc41835441)

[e) Step 5. Heat conduction 20](#_Toc41835442)

[f) Step 6 Darcy and heat conduction coupling 20](#_Toc41835443)

[g) Step9 21](#_Toc41835444)

[III. Using SINK boundary conditions 21](#_Toc41835445)

[A. Basic sink formulation 21](#_Toc41835446)

[B. Sink tests (https://www.mooseframework.org/modules/porous\_flow/tests/sinks/sinks\_tests.html#! ) 22](#_Toc41835447)

[1. Test 1 22](#_Toc41835448)

[2. Gaussian moving heat source 22](#_Toc41835449)

[IV. Adding heat source to the phase-field model of sintering 23](#_Toc41835450)

[References 24](#_Toc41835451)

# List of Figures

[Figure 1 The output ov Sudipta code for sintering two solid particles. 4](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834654)

[Figure 2 Three sintered particles. 4](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834655)

[Figure 3 Displacement (top) and stress (bottom) for three sintered particles. Notice large changes of the values for both quantities (units?) 6](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834656)

[Figure 4. The outcome of the simulations of three particles sintering (without coupling to the mechanical energy) in 3D. Initial structure on the left final on the right. 7](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834657)

[Figure 5. Results of the simulation of the grain growth model modified to work in 3D 7](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834658)

[Figure 6. The results of simulations of sintering three particles in 3D with modified initial location 8](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834659)

[Figure 7. Random initial location of circles for sintering. We choose circles highlighted red for our simulations. 9](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834660)

[Figure 8. Results of simulations for sintefring of 9 particles coupled to the stress. 9](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834661)

[Figure 9 Periodic BC are imposed in xy directions. In addition, Dirichlet BC for the strain are imposed at the top and bottom y axis. 10](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834662)

[Figure 10. Periodic BC in XY direction. No Dirichlet BC. 10](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834663)

[Figure 11. Results of the simulations for G=0.2 m/s2 and D = 2 kg/m3. 11](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834664)

[Figure 12. (left) The results of the simuations of concentration for G=2 m/s2; D=2 kg/m3; =-3 m/m; and = 0 m/m. (right) Stress . 11](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834665)

[*Figure 13 Dimensional and nondimensionalized mobilities used in the parameteric studies in (Biswas et al., 2016)* 13](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834666)

[Figure 14 Tungsten diffusion coefficients (C. Minkwitz, 1997) 14](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834667)

[Figure 15 Surface energy, grain boundary energy and ideal work of grain boundary decohesion for tungsten with different impurities. (Grujicic et al., 1999) 14](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834668)

[Figure 16 Full sintering of a set of spherical particles. 16](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834669)

[Figure 17 Table of material constants and model parameters for diamond-nickel sintering. 17](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834670)

[Figure 18 Mesh and results of simulation of the heat conduction in a circular domain. 17](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834671)

[Figure 19 Solution of Step1.i 19](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834672)

[Figure 20 Solution of Step2.i 19](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834673)

[Figure 21 Results of simulatins at Step9.i 21](file:///E:\boundmetaldeposition\pptx\PH_REPORT.docx#_Toc41834674)

# List of Tables

[Table 1 Script for thermomechanical coupling in a circular domain. 18](#_Toc41834675)

[Table 2 Additions at step 4 19](#_Toc41834676)

[Table 3 Heat conduction: step5c\_outflow.i (check the steady version Step5a\_steady.i) 20](#_Toc41834677)

[Table 4 Step6\_coupled.i 20](#_Toc41834678)

[Table 5 Step9.i 21](#_Toc41834679)

[Table 6. Script for Test1 of sink 22](#_Toc41834680)

[Table 7 Moving Gaussiaan source for pure heat conduction in generic material 22](#_Toc41834681)

# Model development in MOOSE

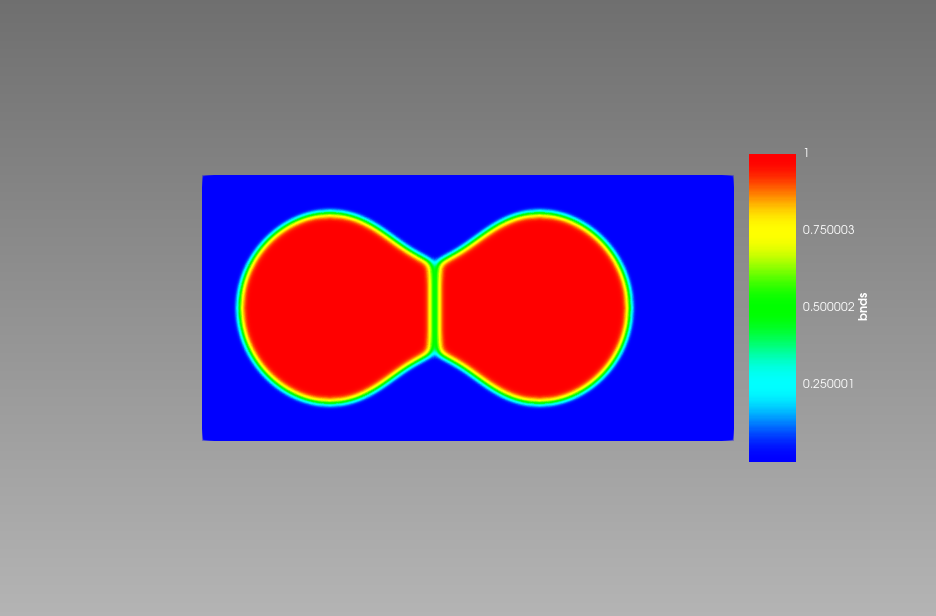
We use Idaho National Lab’s (INL’s) Multiphysics Object-Oriented Simulation Environment

(MOOSE) for implementing the current phase field model.

We begin model development by cloning MOOSE script [sintering\_test1\_isomob.i](https://github.com/SudiptaBiswas/Crow/blob/devel/test/tests/sintering_test1_isomob.i) written by Sudipta Biswas (<https://github.com/SudiptaBiswas/Crow/tree/devel>).

The original code models sintering of two particles that is not coupled to neither solid mechanics nor the thermal conduction. However, it was demonstrated in earlier publications ([1], [2], [4], [7], [8], [9], [10]) that this code can be extended to couple phase field equations to mechanical stress and thermal conduction.

Figure 1 The output ov Sudipta code for sintering two solid particles.

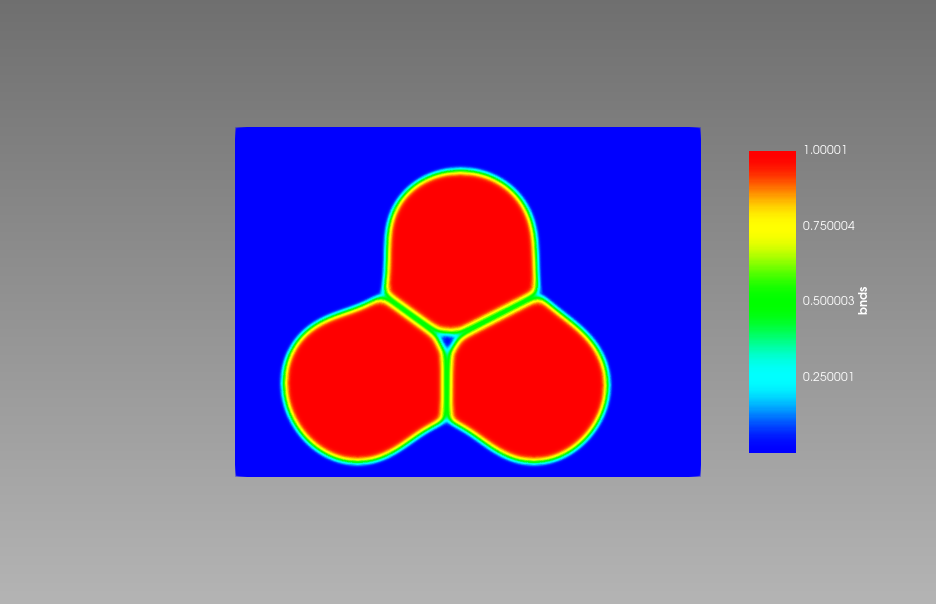


## Adding one more particle in 2D ([sint\_t1\_isomob\_3gr.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/3gr/sint_t1_isomob_3gr.i))

To add one more particle we have changed global parameter op\_num to 3

  op\_num = 3.0

and added new variable gr2 at every location where we see variables gr0 and gr1. For example, in the module IC’s we have added.



We have also changed coordinates of all particles to ensure initial contact.

Figure 2 Three sintered particles.

  [./ic\_gr2]

    int\_width = 2.0

    x1 = 25.2

    y1 = 8.0

    radius = 7.0

    outvalue = 0.0

    variable = gr2

    invalue = 1.0

    type = SmoothCircleIC

  [../]

## Coupling to elastic stress ([sint\_t1\_isom\_3gr\_el\_3S.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/3gr_Fc_Fe/sint_t1_isom_3gr_el_3S.i))

A few ways are available to couple the model to mechanical stress. Here we use one of the simplest methods based on extension of the following example: [Conserved3d.i](https://github.com/luchinsky/Crow/blob/develD/examples/combined/phase-field-mechanics/Conserved/Conserved3d.i) .

The main changes include:

* Add displacement to global parameters: displacements = 'disp\_x disp\_y'
* Add displacements to the definition of variables

  [./disp\_x]

    order = FIRST

    family = LAGRANGE

  [../]

  [./disp\_y]

    order = FIRST

    family = LAGRANGE

  [../]

* Add auxiliary variables (note that the IC’s did not change)

  [./sigma11\_aux]

    order = CONSTANT

    family = MONOMIAL

  [../]

  [./sigma22\_aux]

    order = CONSTANT

    family = MONOMIAL

  [../]

* Add TensorMechanics to the Kernels

  [./TensorMechanics]

    displacements = 'disp\_x disp\_y'

  [../]

* Add stress calculations to auxiliary kernals

  [./matl\_sigma11]

    type = RankTwoAux

    rank\_two\_tensor = stress

    index\_i = 0

    index\_j = 0

    variable = sigma11\_aux

  [../]

  [./matl\_sigma22]

    type = RankTwoAux

    rank\_two\_tensor = stress

    index\_i = 1

    index\_j = 1

    variable = sigma22\_aux

  [../]

* Add boundary conditions

  [./bottom\_y]

    type = DirichletBC

    variable = disp\_y

    boundary = 'bottom'

    value = 0

  [../]

  [./top\_y]

    type = DirichletBC

    variable = disp\_y

    boundary = 'top'

    value = -5

  [../]

  [./left\_x]

    type = DirichletBC

    variable = disp\_x

    boundary = 'left'

    value = 0

  [../]

* Add materials to calculate Elastic Free Energy (note that at this stage it was not possible to exclude any of this materials)

  [./elasticity\_tensor]

    type = ComputeElasticityTensor

    block = 0

    # lambda, mu values

    C\_ijkl = '7 7'

    # Stiffness tensor is created from lambda=7, mu=7 using symmetric\_isotropic fill method

    fill\_method = symmetric\_isotropic    # See RankFourTensor.h for details on fill methods

  [../]

  [./stress]

    type = ComputeLinearElasticStress

    block = 0

  [../]

    [./var\_dependence]

    type = DerivativeParsedMaterial

    block = 0

    function = 0.2\*c

    args = c

    f\_name = var\_dep

    enable\_jit = true

    derivative\_order = 2

  [../]

  [./eigenstrain]

    type = ComputeVariableEigenstrain

    block = 0

    eigen\_base = '1 1 1 0 0 0'

    prefactor = var\_dep

    args = 'c'

    eigenstrain\_name = eigenstrain

  [../]

  [./strain]

    type = ComputeSmallStrain

    block = 0

    displacements = 'disp\_x disp\_y'

    eigenstrain\_names = eigenstrain

  [../]

  [./elastic\_free\_energy]

    type = ElasticEnergyMaterial

    f\_name = Fe

    block = 0

    args = 'c'

    derivative\_order = 2

  [../]

* Use SinteringFreeEnergy as partial chemical free energy

  [./chemical\_free\_energy]

    type = SinteringFreeEnergy

    block = 0

    c = c

    v = 'gr0 gr1 gr2'

    f\_name = Fc

    derivative\_order = 2

  [../]

* Add sum of two free energies: Fc and Fe (chemical and elastic)

  [./free\_energy]

    type = DerivativeSumMaterial

    block = 0

    f\_name = F

    sum\_materials = 'Fc Fe'

    args = 'c'

    derivative\_order = 2

  [../]

* Add elastic and chemical free energy to the output

  [./el\_free\_en]

    type = ElementIntegralMaterialProperty

    mat\_prop = Fe

  [../]

  [./ch\_free\_en]

    type = ElementIntegralMaterialProperty

    mat\_prop = Fc

  [../]

The output for displacement and stress of three sintered particles is shown in Figure 3. The important highlight at this stage is that the observed changes are significant and these changes follow the sintering dynamics.

## Increase dimension to 3D ([sint\_3gr\_3.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/3gr_2/sint_3gr_3.i))

This extension does not work properly yet. The following changes have been introduced

* Mesh

[Mesh]

  type = GeneratedMesh

  dim = 3

  nx = 16

  ny = 16

  nz = 8

  xmin = 0.0

  xmax = 40.0

  ymin = 0.0

  ymax = 30.0

  zmin = 0

  zmax = 30.0

  elem\_type = HEX8

[]

However, the outcome of the simulations shown in Figure 4 reveals the fact that the particles grow in two dimensions.

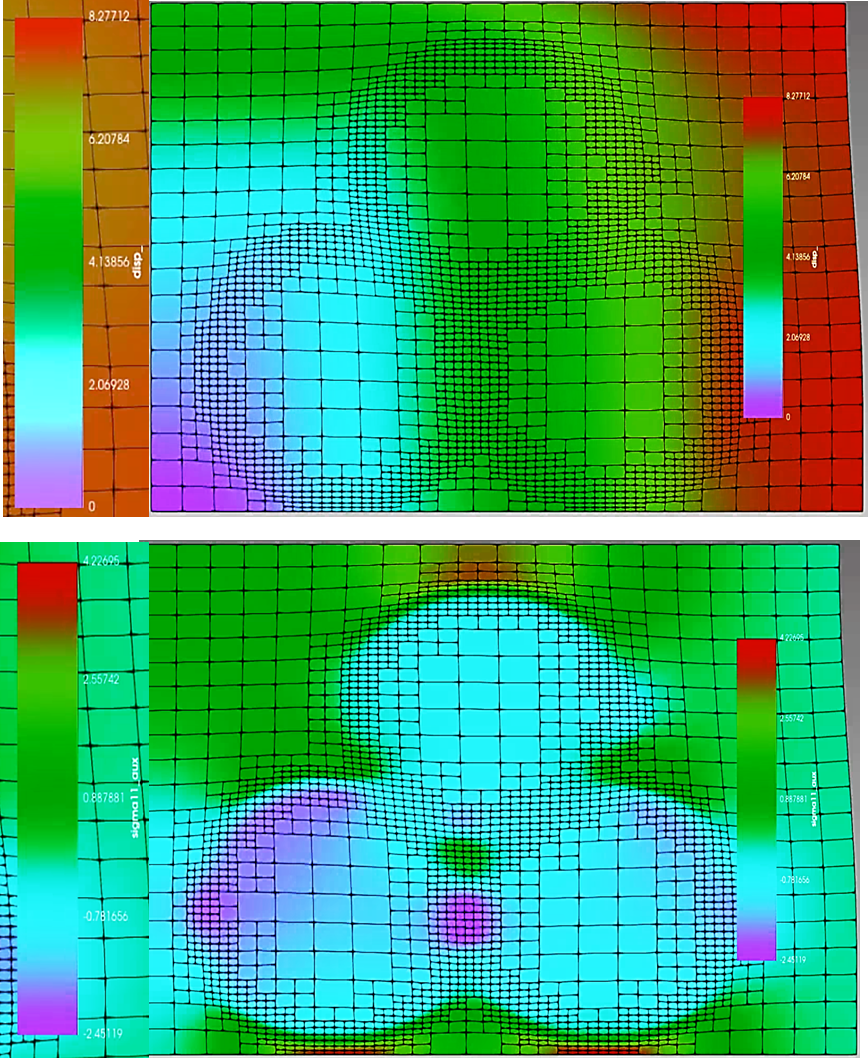


Figure Displacement (top) and stress (bottom) for three sintered particles. Notice large changes of the values for both quantities (units?)

To analyze the issue the following tests were performed. We check a few 3D tests from MOOSE

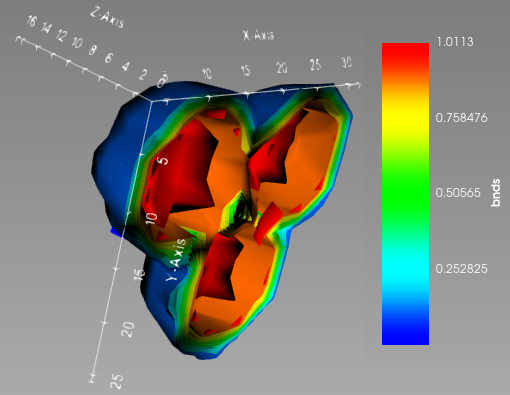


Figure 4. The outcome of the simulations of three particles sintering (without coupling to the mechanical energy) in 3D. Initial structure on the left final on the right.

### Grain growth 3D ([grain\_growth\_3D\_random.i](https://github.com/luchinsky/Crow/blob/develD/examples/phase-field/PFtests/grain_growth/grain_growth_3D_random.i)).

# This simulation predicts GB migration of a 2D copper polycrystal with 100 grains represented with 18 order parameters

# Mesh adaptivity and time step adaptivity are used

# An AuxVariable is used to calculate the grain boundary locations

# Postprocessors are used to record time step and the number of grains

We modify this example to work in 3D.

[Mesh]

  # Mesh block.  Meshes can be read in or automatically generated

  type = GeneratedMesh

  dim = 3 # Problem dimension

  nx = 10 # Number of elements in the x-direction

  ny = 10 # Number of elements in the y-direction

  nz = 10

  xmin = 0    # minimum x-coordinate of the mesh

  xmax = 1000 # maximum x-coordinate of the mesh

  ymin = 0    # minimum y-coordinate of the mesh

  ymax = 1000 # maximum y-coordinate of the mesh

  zmin = 0

  zmax = 1000

  uniform\_refine = 1 # Initial uniform refinement of the mesh

  parallel\_type = distributed

[]

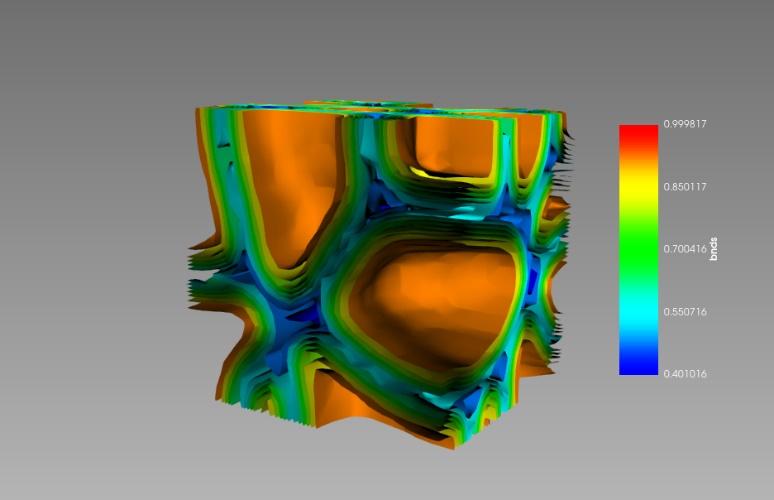


Figure 5. Results of the simulation of the grain growth model modified to work in 3D

The results of the simulations are shown in Figure 5.

### Changing [Conserved.i](https://github.com/luchinsky/Crow/blob/develD/examples/combined/phase-field-mechanics/Conserved/Conserved.i) to 3D

Similarly, the MOOSE example [Conserved\_3d.i](https://github.com/luchinsky/Crow/blob/develD/examples/phase-field/phase_field-mechanics/Conserved/Conserved_3d.i) was modified to work in 3D.

# Example 1.

# Illustrating the coupling between chemical and mechanical (elastic) driving forces.

# An oversized precipitate deforms under a uniaxial compressive stress

# Check the file below for comments and suggestions for parameter modifications.

By changing mesh options

[Mesh]

  type = GeneratedMesh

  dim = 3

  nx = 16

  ny = 16

  nz = 8

  xmin = 0

  xmax = 50

  ymin = 0

  ymax = 50

  zmin = 0

  zmax = 50

  elem\_type = HEX8

[]

### Changing location of the particles in sint\_3gr\_3

We have changed the location of the particles in the original code discussed in Section C above

The new code is [sint\_3gr\_3d\_1.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/3gr_2/sint_3gr_3d_1.i) . In the new code only the initial conditions have been changed

|  |  |
| --- | --- |
| [ICs]    [./ic\_gr2]      int\_width = 2.0      x1 = 25.2      y1 = 8.0      z1 = 12.0      radius = 7.0      outvalue = 0.0      variable = gr2      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr1]      int\_width = 2.0      x1 = 19.367      y1 = 19.488      z1 = 18.0      radius = 6.0      outvalue = 0.0      variable = gr1      invalue = 1.0      type = SmoothCircleIC    [../] | [./ic\_gr0]      int\_width = 2.0      x1 = 11.0      y1 = 8.0      z1 = 14.0      radius = 7.4      outvalue = 0.0      variable = gr0      invalue = 1.0      type = SmoothCircleIC    [../]    [./multip]      x\_positions = '11.0 19.367  25.2'      int\_width = 2.0      z\_positions = '14 18 12'      y\_positions = '8.0 19.488 8.0 '      radii = '7.4 6.0 7.0'      3D\_spheres = false      outvalue = 0.001      variable = c      invalue = 0.999      type = SpecifiedSmoothCircleIC      block = 0    [../] |

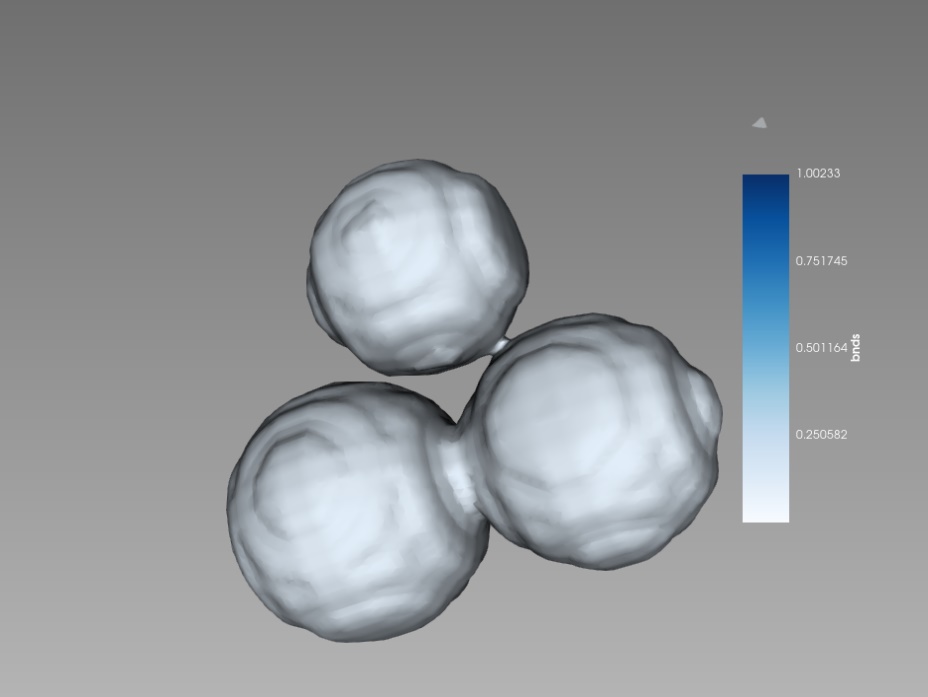
We shifted the location of the particles to the center of the cell. The result of simulations is shown in the Figure 6.

We note that it takes quite some time to simulate three particles sintering in this code. So, for now we return to the 2D simulations.

## 3D model with more than 3 particles

We use as a prototype the model [sint\_t1\_isom\_3gr\_el\_3S.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/3gr_Fc_Fe/sint_t1_isom_3gr_el_3S.i) developed in Section B. The changes he changes in the script are the following, cf changes in Section B.

Figure 6. The results of simulations of sintering three particles in 3D with modified initial location



|  |  |  |
| --- | --- | --- |
| type = GeneratedMesh    dim = 2    nx = 60    ny = 60    #   nz = 0    xmin = 0.0    xmax = 40.0    ymin = 0.0    ymax = 40.0    # zmax = 0    elem\_type = QUAD4  []  **[GlobalParams]**    displacements = 'disp\_x disp\_y'    var\_name\_base = gr    op\_num = 9.0    int\_width = 1.0    #en\_ratio = 1  []  **[Variables]**  **[./c]**      #scaling = 10    [../]  **[./w]**    [../]  **[./PolycrystalVariables]**    [../]  **[./disp\_x]**      order = FIRST      family = LAGRANGE    [../]  **[./disp\_y]**      order = FIRST      family = LAGRANGE    [../]  **[ICs]**  **[./ic\_gr8]**      int\_width = 2.0      x1 = 18.9158      y1 = 18.2981      radius = 3.895      outvalue = 0.0      variable = gr8      invalue = 1.0      type = SmoothCircleIC    [../]  **[./ic\_gr7]**      int\_width = 2.0 | x1 = 10.6328      y1 = 29.4843      radius = 3.75      outvalue = 0.0      variable = gr7      invalue = 1.0      type = SmoothCircleIC    [../]  **[./ic\_gr6]**      int\_width = 2.0      x1 = 21.7174      y1 = 15.3236      radius = 3.75      outvalue = 0.0      variable = gr6      invalue = 1.0      type = SmoothCircleIC    [../]  **[./ic\_gr5]**      int\_width = 2.0      x1 = 20.0109      y1 = 30.5594      radius = 4.32      outvalue = 0.0      variable = gr5      invalue = 1.0      type = SmoothCircleIC    [../]  **[./ic\_gr4]**      int\_width = 2.0      x1 = 27.8199      y1 = 28.1836      radius = 3.375      outvalue = 0.0      variable = gr4      invalue = 1.0      type = SmoothCircleIC    [../]  **[./ic\_gr3]**      int\_width = 2.0      x1 = 22.0109      y1 = 22.7441      radius = 3.375      outvalue = 0.0      variable = gr3      invalue = 1.0      type = SmoothCircleIC    [../] | **[./ic\_g2]**      int\_width = 2.0      x1 = 13.5818      y1 = 17.6532      radius = 3.375      outvalue = 0.0      variable = gr2      invalue = 1.0      type = SmoothCircleIC    [../]  **[./ic\_gr1]**      int\_width = 2.0      x1 = 8.592      y1 = 22.4133      radius = 3.25      outvalue = 0.0      variable = gr1      invalue = 1.0      type = SmoothCircleIC    [../]  **[./ic\_gr0]**      int\_width = 2.0      x1 = 15.6702      y1 = 24.1294      radius = 3.25      outvalue = 0.0      variable = gr0      invalue = 1.0      type = SmoothCircleIC    [../]  **[./multip]**      x\_positions = '28.9158 10.6328  21.7174 20.0109 27.8199 22.9458 13.5818 8.592 15.6702'      y\_positions = '18.2981 29.4843  15.3236 30.5594 28.1836 22.7441 17.6532 22.4133 24.1294'      z\_positions = '0 0 0 0 0 0 0 0 0'      radii = '3.875  3.75  3.75  4.325 3.5 3.375 3.375 3.25  3.25'      int\_width = 2.0      3D\_spheres = false      outvalue = 0.001      variable = c      invalue = 0.999      type = SpecifiedSmoothCircleIC      block = 0    [../]  [] |

To find initial position of the particles we use matlab code bubblebath.m to generate initial random location of particles.

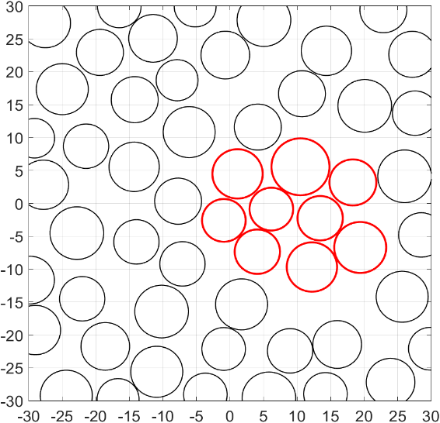
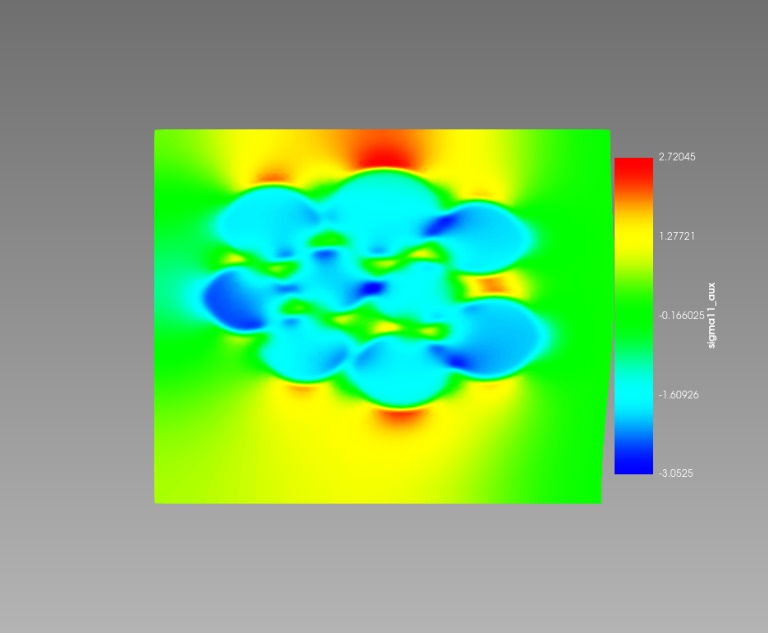
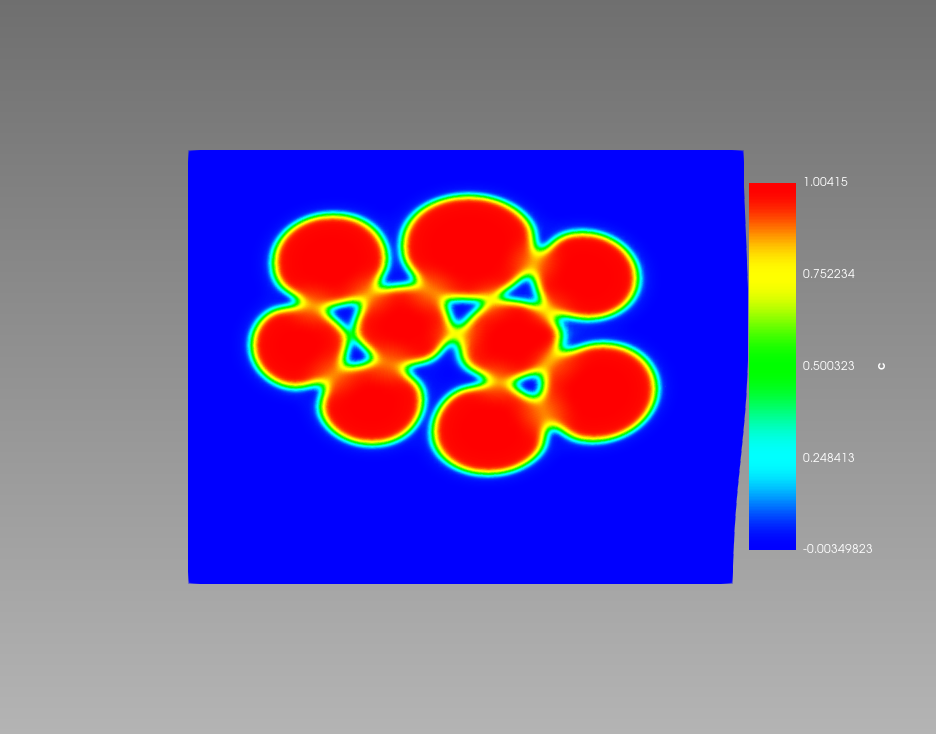


Figure 7. Random initial location of circles for sintering. We choose circles highlighted red for our simulations.

The outcome of simulations is shown in the Figure 8.

Figure 8. Results of simulations for sintefring of 9 particles coupled to the stress.



## Adding periodic boundary conditions

Periodic BC ([sint1\_isom\_9gr\_el\_2d\_P.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/9gr_2d/sint1_isom_9gr_el_2d_P.i)) and ([sint1\_isom\_9gr\_el\_2d\_P1.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/9gr_2d/sint1_isom_9gr_el_2d_P1.i))

MOOSE provides built-in support for specifying periodic boundary conditions. The [periodic\_bc.i](https://github.com/idaholab/moose/blob/next/examples/ex04_bcs/periodic_bc.i) input file demonstrates this functionality while also taking advantage of MOOSE's ability for parsing user-specified analytical functions from input files. Parsed functions can be used to do many things in input files and are discussed in more detail in [Example 13](https://mooseframework.inl.gov/examples/ex13_functions.html).

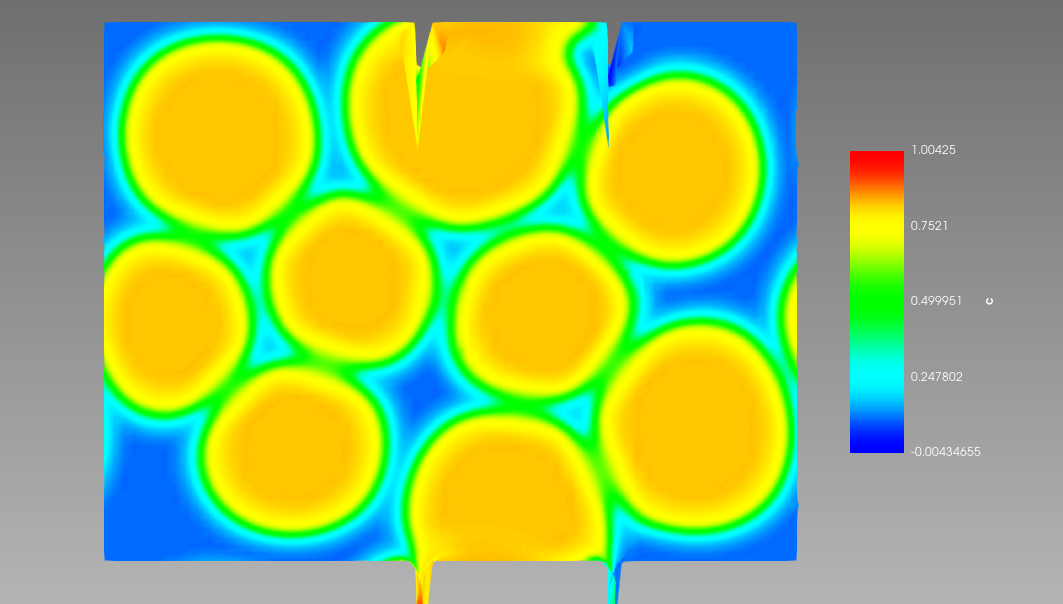


Figure 9 Periodic BC are imposed in xy directions. In addition, Dirichlet BC for the strain are imposed at the top and bottom y axis.

In the first example we impose periodic BC in XY directions. In addition, Dirichlet BC for the strain are imposed at the top and bottom y axis. The simulation results are shown in Figure 9.

The artifacts observed at the top and bottom are, I believe, due to incompatibility of the periodic and Dirichlet BC. The corrected results (no Dirichlet BC) are shown in Figure 10.

[BCs]

# Boundary Condition block

  [./Periodic]

    [./top\_bottom]

      auto\_direction = 'x y' # Makes problem periodic in the x and y directions

    [../]

  [../]

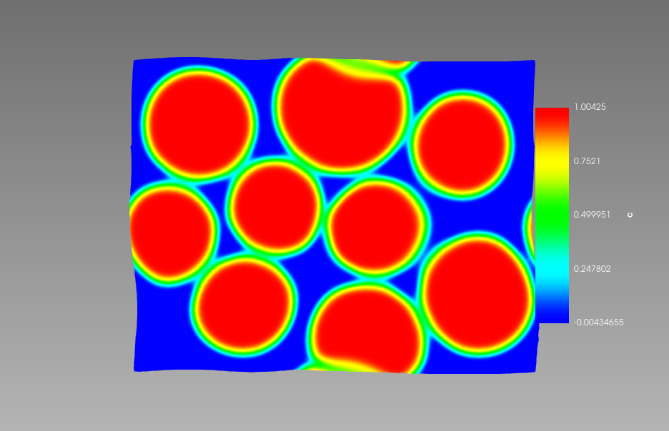
## Gravity ([sint1\_isom\_9gr\_el\_2d\_PG\_02\_m3\_0.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/9gr_2d/G_02_yt_m3_yb_0/sint1_isom_9gr_el_2d_PG_02_m3_0.i))

[sint1\_isom\_9gr\_el\_2d\_PG\_2\_yt\_m3\_yb\_0.i](https://github.com/luchinsky/Crow/blob/develD/test/tests/9gr_2d/G_2_ytm3_yb_0/sint1_isom_9gr_el_2d_PG_2_yt_m3_yb_0.i)

To add gravity we need to

* remove periodic BC and in Y direction
* add gravity value in that direction
* restore Dirichlet BC in this direction (otherwise there is no convergence so far)

Figure 10. Periodic BC in XY direction. No Dirichlet BC.



[BCs]

# Boundary Condition block

  [./Periodic]

    [./top\_bottom]

      auto\_direction = 'x y' # Makes problem periodic in the x and y directions

    [../]

  [../]

   [./bottom\_y]

     type = *DirichletBC*

     variable = disp\_y

     boundary = 'bottom'

     value = 0

   [../]

   [./top\_y]

     type = *DirichletBC*

     variable = disp\_y

     boundary = 'top'

     # prescribed displacement

     # -5 will result in a compressive stress

     #  5 will result in a tensile stress

     value = -3

[]

[Kernels]

  [./TensorMechanics]

    displacements = 'disp\_x disp\_y'

  [../]

  [./gravity\_y]

    type = *Gravity*

    variable = disp\_y

    # value = -0.2 # 1.81

    value = -2.0 # 1.81

  [../]

[]

There is one more important consideration we have to specify density as a new material property

[Materials]

  [./density]

    type = GenericConstantMaterial

    prop\_names = density

    prop\_values = 2.0387

  [../]

[]

Figure 11. Results of the simulations for G=0.2 m/s2 and D = 2 kg/m3.

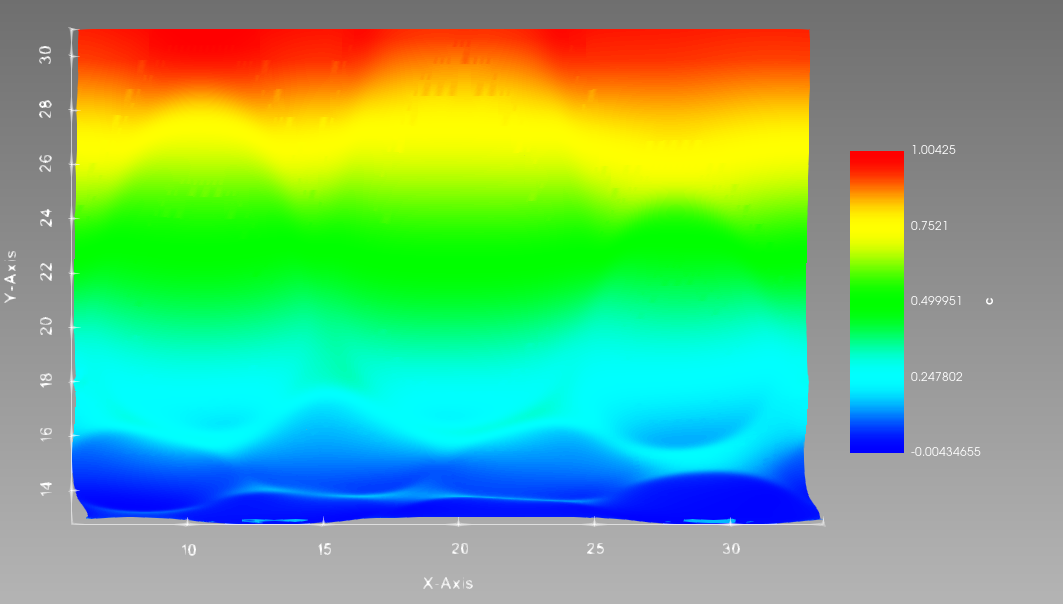
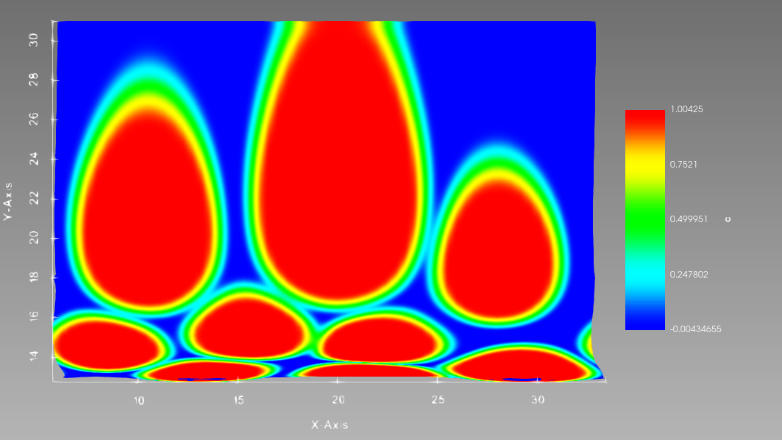
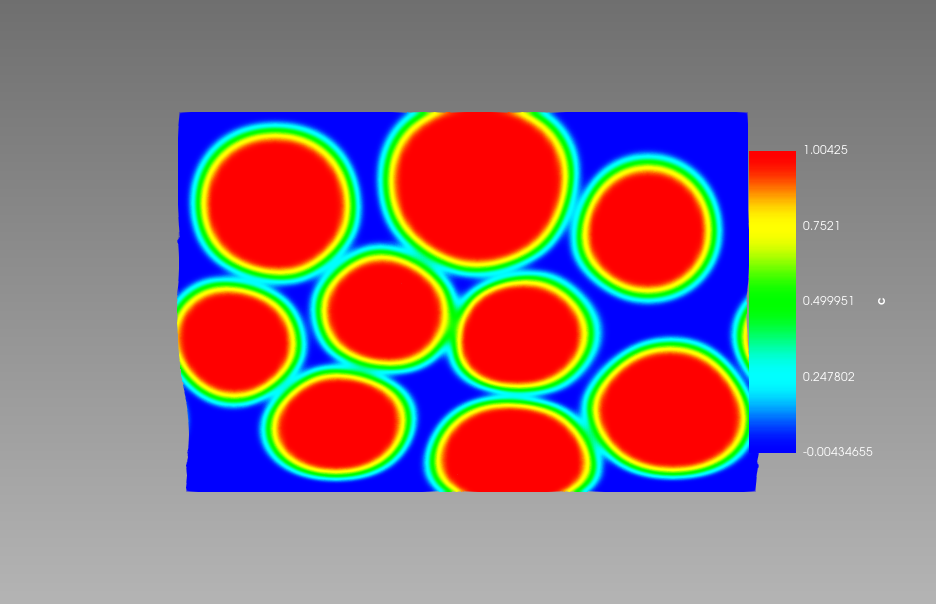


Figure 12. (left) The results of the simuations of concentration for G=2 m/s2; D=2 kg/m3; =-3 m/m; and = 0 m/m. (right) Stress .

The importance of the parameter (variable) values is apparent in this example. In the first example G=2 m/s2 and density is D=2 kg/m3. The results of simulations are shown in Figure 11. It is clear, that the density is substantially underestimated. For metals it is expected to be of the order 5 – 10 kg/m3. At the same time the dimensions presented in the figure correspond to mm or even mkm.

The results of the simulations for one more set of parameter values are shown in

For reduced gravity the deformation of the particles is (as expected) substantially smaller than in previous case.

However, the proper choice of the parameter values must be done self-consistently to reflect the dimensions and units of the problem.

## UNITS

Handling Units in MOOSE

### [Phase field](https://mooseframework.inl.gov/old/wiki/PhysicsModules/PhaseField/PhaseFieldUnits/)

There is no inherit unit system in MOOSE. Thus, the units of the phase field equations are set by the user when they define a model. Specifically, the units are set by the local free energy density and the κ and mobility parameters (L and M). The units in all these terms must be consistent. Additional energy sources, such as the elastic energy, also must have consistent units. In the phase field module, all of these values are created using Material objects. Thus, the units of your system are not set by the kernels but rather by the materials.

One useful practice is to create your material objects to take SI units as input parameters. Then use length\_scale, time\_scale, and energy\_scale input parameters to convert the actual units of the problem. As an example of this, see the PFParamsPolyFreeEnergy material, where the input file block looks like

[./Copper]

  type = PFParamsPolyFreeEnergy

  block = 0

  c = c

  T = 1000 # K

  int\_width = 30.0

  length\_scale = 1.0e-9

  time\_scale = 1.0e-9

  D0 = 3.1e-5 # m^2/s, from Brown1980

  Em = 0.71 # in eV, from Balluffi1978 Table 2

  Ef = 1.28 # in eV, from Balluffi1978 Table 2

  surface\_energy = 0.7 # J/m^2

[../]

The Cahn-Hilliard equation after the variational derivative takes the form









The Allen-Cahn equation after the variational derivative takes the form 



where and are gradient energy coefficients with the units , describes any additional sources of energy in the system, such as deformation or electrostatic energy, with units of the units of are and the units of are . Note that some models include the in the mobility term, such that it has units of

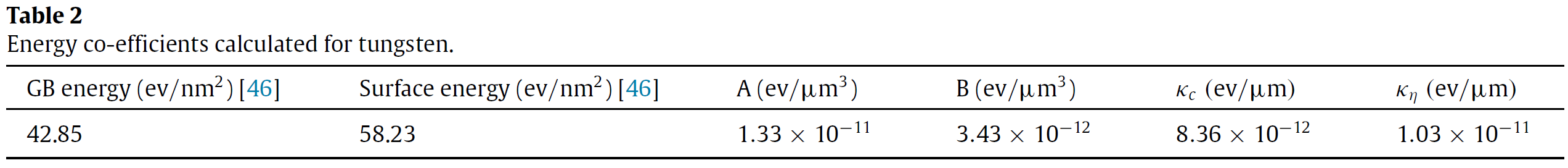
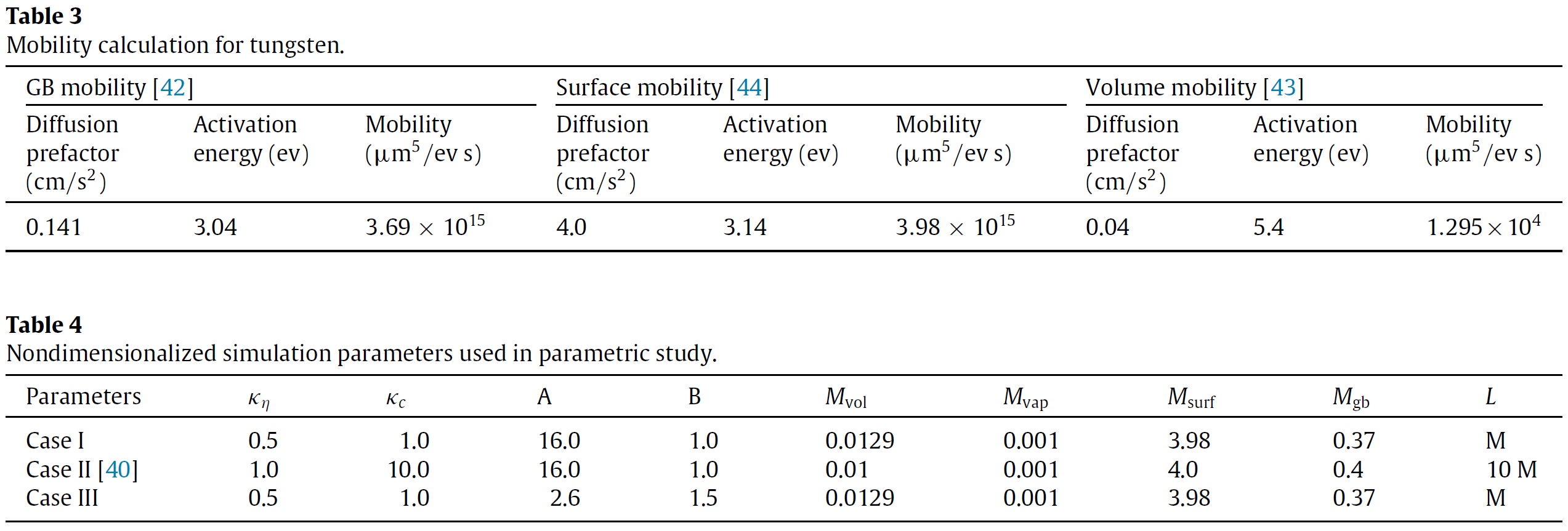


### A, B, M, L and coefficients

With units of the units of are and the units of are , and are gradient energy coefficients with the units .



Mobility coefficients () are estimated from corresponding diffusion coefficients as



*Figure 13 Dimensional and nondimensionalized mobilities used in the parameteric studies in (Biswas et al., 2016)*

and are the surface and the grain boundary energy, is the interface width or the grain boundary thickness. are normalization constants.

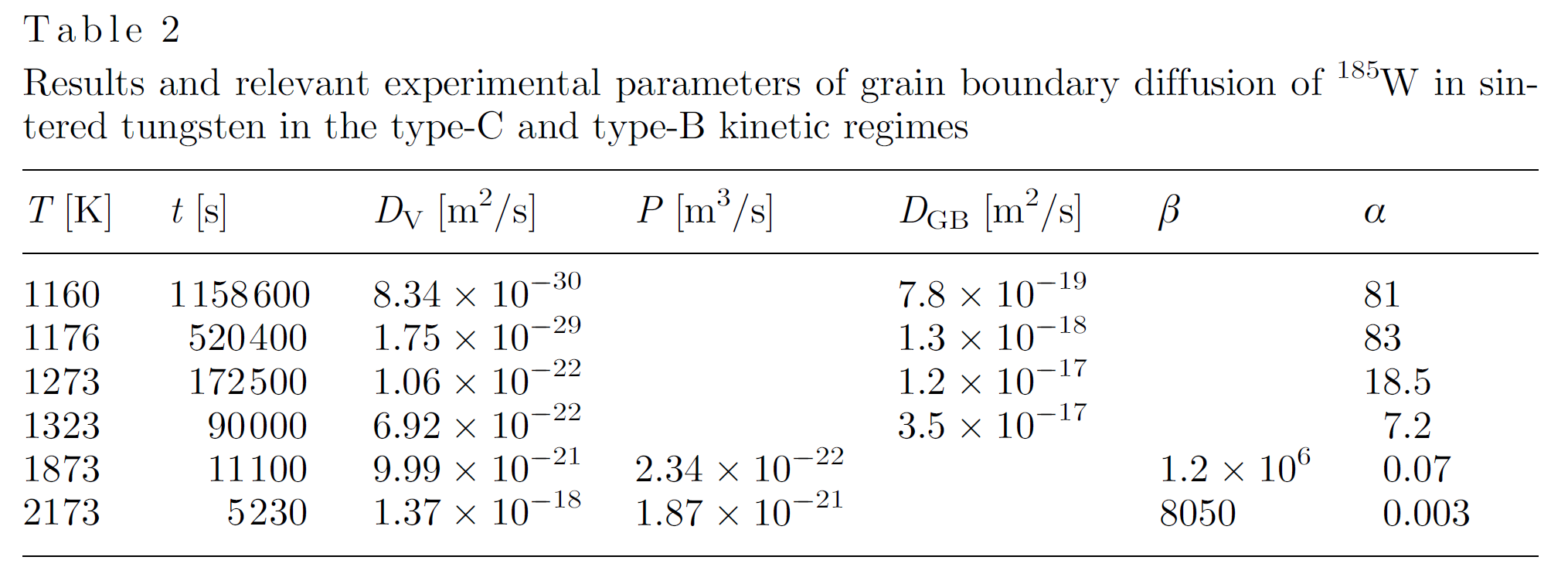
UNITS. Molar volume of tungsten is

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 kcal/mol = | 0.04336 eV |  | 1 kT unit = | 0.5922 kcal/mol |  |
|  | 349.75 cm-1 |  | (for T = 298 K) | 0.02568 eV |  |
|  | 1.689 kT |  |  | 207.1 cm-1 |  |
|  | 4.184 kJ/mol |  |  | 2.476 kJ/mol |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 1 eV = | 23.06 kcal/mol |  | 1 Hartree = | 1 atomic unit |  |
|  | 8065.6 cm-1 |  |  | 627.51 kcal/mol |  |
|  | 38.94 kT |  |  | 27.2114 eV |  |
|  | 96.49 kJ/mol |  |  | 1059.70 kT |  |
|  |  |  |  |  |  |

Consider one example: , , , .

Even simper example: while dimensionless value is ~1.

Figure 14 Tungsten diffusion coefficients (C. Minkwitz, 1997)



In the code these coefficients are defined in material properties

  [./constant\_mat]

    type = *GenericConstantMaterial*

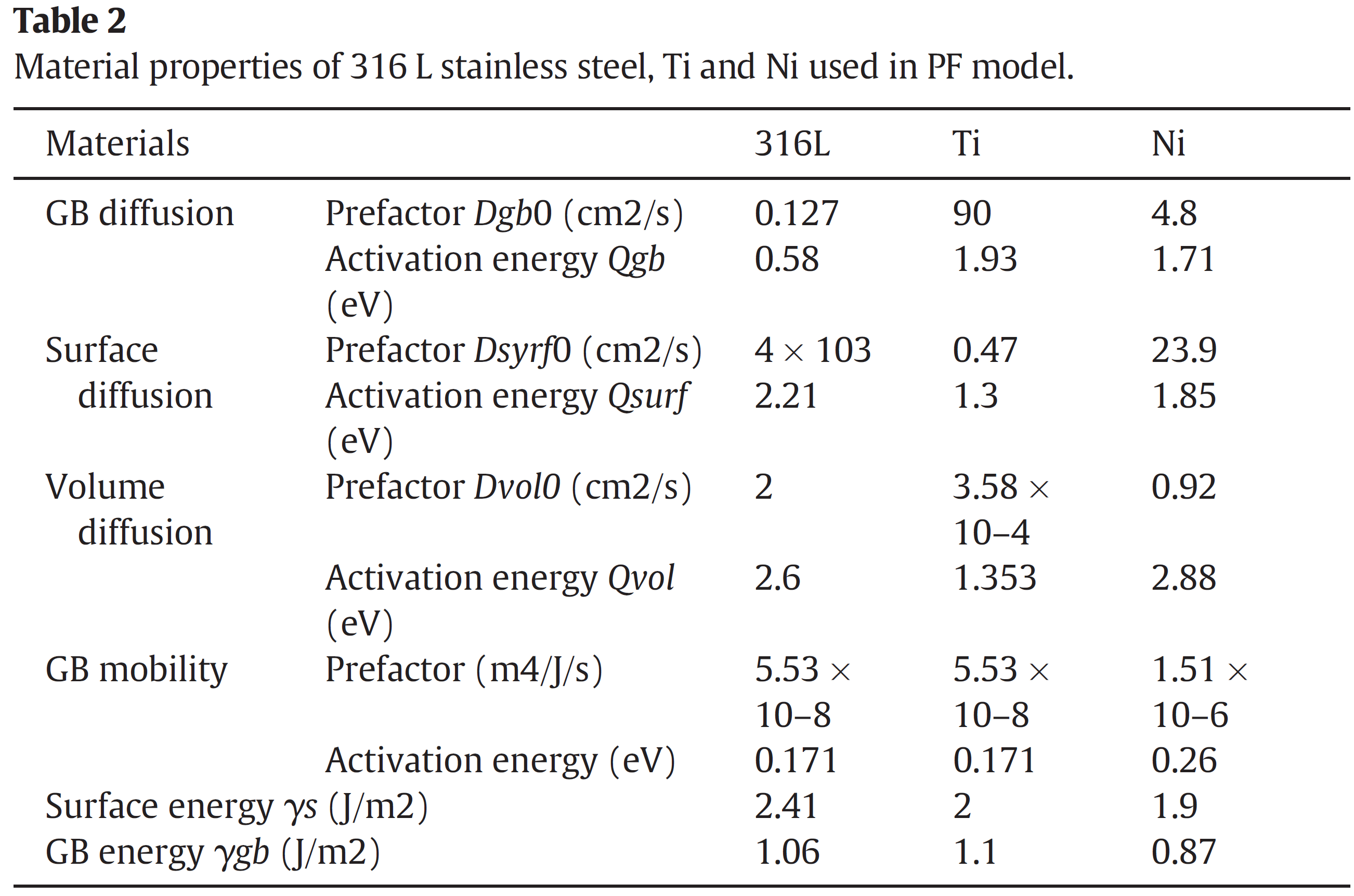
    block = 0

    prop\_names = '  A    B   L   kappa\_op kappa\_c'

    prop\_values = '16.0 1.0 1.0  0.5      1.0    '

  [../]

Figure 15 Parameters for Ti and Ni from [10]



### Parameters from [10]

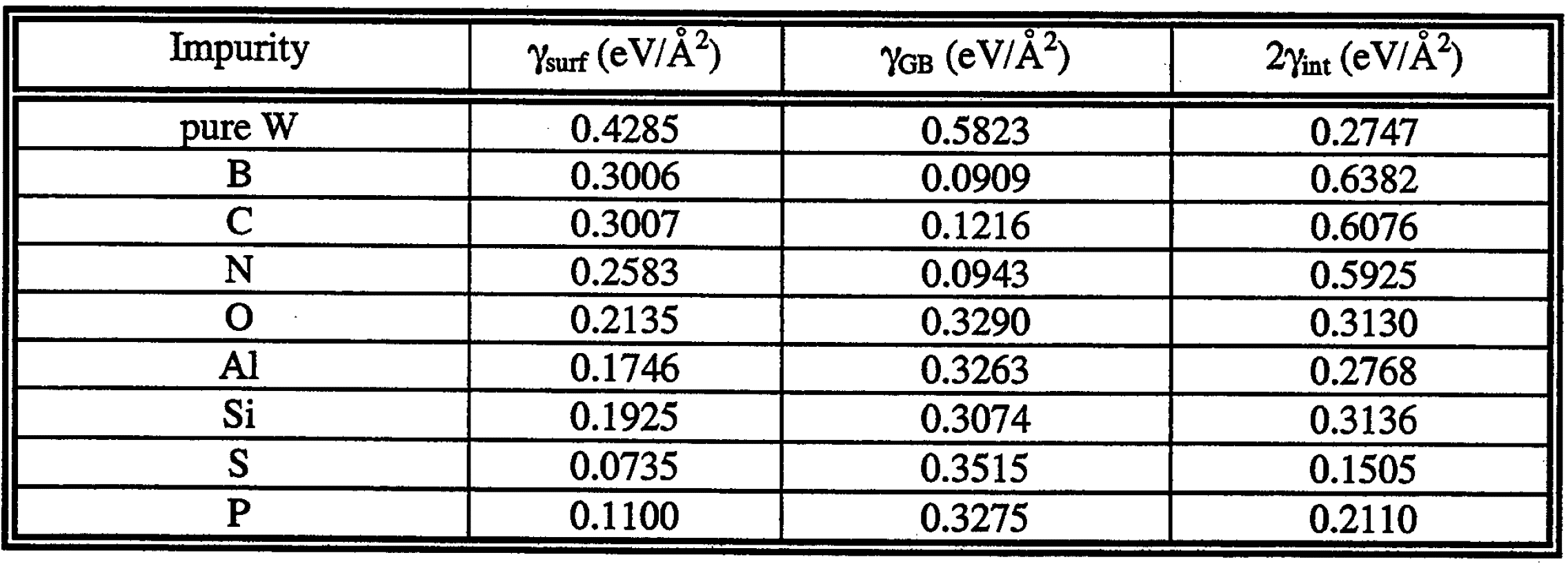


Figure 16 Surface energy, grain boundary energy and ideal work of grain boundary decohesion for tungsten with different impurities. (Grujicic et al., 1999)

### [Mechanics](https://mooseframework.inl.gov/old/wiki/PhysicsModules/TensorMechanics/)

The strong form of the governing equation on the domain and boundary can be stated as follows:





where is the Cauchy stress tensor, is an additional source of stress (such as pore pressure), is the displacement vector, is the body force per unit volume(?), is the unit normal to the boundary, is the prescribed displacement on the boundary and is the prescribed traction on the boundary. The weak form of the residual equation is expressed as:



where and represent volume and boundary integrals, respectively. The solution of the residual equation with Newton's method requires the Jacobian of the residual equation, which can be expressed as (ignoring boundary terms)



assuming is independent of the strain.

The material stress response is described by the constitutive model, where the stress is determined as a function of the strain, i.e. where is the strain and is a stress free strain. For example, in linear elasticity (only valid for small strains), the material response is linear, i.e. The tensor mechanics system can handle linear elasticity and finite strain mechanics, including both elasticity and plasticity.

### Modifying code

First, we modify the dimensions of the mesh to make it suitable for tens of mkm size of the particles

[Mesh]

  type = GeneratedMesh

  dim = 2

  nx = 60

  ny = 60

  #   nz = 0

  xmin = 60.0

  xmax = 330.0

  ymin = 130.0

  ymax = 340.0

  # zmax = 0

  elem\_type = QUAD4

[]

Next, we have to modify the initial conditions to match the mesh scale

|  |  |  |  |
| --- | --- | --- | --- |
| [ICs]    [./ic\_gr8]      int\_width = 2.0      x1 = 280.9158      y1 = 180.2981      radius = 30.895      outvalue = 0.0      variable = gr8      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr7]      int\_width = 2.0      x1 = 100.6328      y1 = 290.4843      radius = 30.75      outvalue = 0.0      variable = gr7      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr6]      int\_width = 2.0      x1 = 21.7174      y1 = 15.3236      radius = 3.75      outvalue = 0.0      variable = gr6      invalue = 1.0      type = SmoothCircleIC    [../] | [./ic\_gr5]      int\_width = 2.0      x1 = 200.0109      y1 = 300.5594      radius = 40.32      outvalue = 0.0      variable = gr5      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr4]      int\_width = 2.0      x1 = 270.8199      y1 = 280.1836      radius = 30.375      outvalue = 0.0      variable = gr4      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr3]      int\_width = 2.0      x1 = 220.0109      y1 = 220.7441      radius = 30.375      outvalue = 0.0      variable = gr3      invalue = 1.0      type = SmoothCircleIC    [../] | [./ic\_gr5]      int\_width = 2.0      x1 = 200.0109      y1 = 300.5594      radius = 40.32      outvalue = 0.0      variable = gr5      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr4]      int\_width = 2.0      x1 = 270.8199      y1 = 280.1836      radius = 30.375      outvalue = 0.0      variable = gr4      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr3]      int\_width = 2.0      x1 = 220.0109      y1 = 220.7441      radius = 30.375      outvalue = 0.0      variable = gr3      invalue = 1.0      type = SmoothCircleIC    [../] | [./ic\_g2]      int\_width = 2.0      x1 = 130.5818      y1 = 170.6532      radius = 30.375      outvalue = 0.0      variable = gr2      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr1]      int\_width = 2.0      x1 = 80.592      y1 = 220.4133      radius = 30.25      outvalue = 0.0      variable = gr1      invalue = 1.0      type = SmoothCircleIC    [../]    [./ic\_gr0]      int\_width = 2.0      x1 = 150.6702      y1 = 240.1294      radius = 30.25      outvalue = 0.0      variable = gr0      invalue = 1.0      type = SmoothCircleIC    [../] |

  [./multip]

    x\_positions = '280.9158 100.6328    210.7174    200.0109    270.8199    220.9458    130.5818    8.592   150.6702'

    y\_positions = '180.2981 290.4843    150.3236    300.5594    280.1836    220.7441    170.6532    220.4133    240.1294'

    z\_positions = '0 0 0 0 0 0 0 0 0'

    radii = '30.875 30.75   30.75   40.325  30.5    30.375  30.375  30.25   30.25'

    int\_width = 2.0

    3D\_spheres = false

    outvalue = 0.001

    variable = c

    invalue = 0.999

    type = SpecifiedSmoothCircleIC

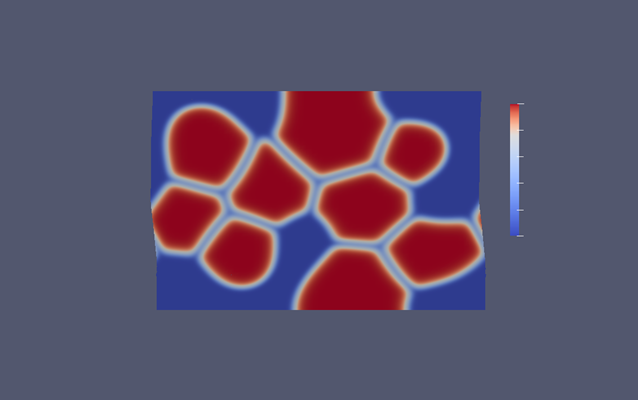
    block = 0

  [../]

[]

The first choice we have to make is gravity units. Should we keep mksm/s2 or should we change to mkm/ms2. This choice must be consistent with the units of energy

Figure 17 Full sintering of a set of spherical particles.



## Example. Spinodal decomposition (<https://mooseframework.inl.gov/old/wiki/MooseTutorials/IronChromiumDecomposition/>)

Consider Cahn-Hillard quation



where *c* is the mole fraction of chromium (unitless), *M*(*c*) is the mobility of chromium (), is the free energy density (),and *κ* is the gradient energy coefficient (). Values for these terms can be looked up in a database or calculated in a thermodynamic software package. while the coefficient *κ* is 8.125×10−16.

In this problem, values were fit to equations which rely on the equation

### Units

The initial condition is given in weight percent, but the equations are given to us on a molar basis. Therefore, we can use the molecular weights of chromium to convert the initial condition from 45 wt% to 46.774 mol% chromium.

In units of meters, the mesh would be entered into MOOSE as 25×10−9 by 25×10−9. However, MOOSE has a built in tolerance that will not allow mesh nodes to be any closer together than 10−6. To get around this we need to change the length scale to units of nanometers. To prevent the values from becoming too large or too small, we will also change the energy scale to units of electron volts. The conversion from meters to nanometers is 109. The conversion from joules to electron volts is 6.24150934×1018. Rather than converting all the values in the table above, we will program these conversions into the input file and do the conversions within the file.

### S1\_testmodel.i ([s1\_testmodel.i](https://github.com/idaholab/moose/blob/devel/modules/phase_field/tutorials/spinodal_decomposition/s1_testmodel.i))

* Important ideas: use explicit dimensions

  [./constants]

    type = *GenericFunctionMaterial*

    block = 0

    prop\_names = 'kappa\_c M'

    prop\_values = '8.125e-16\*6.24150934e+18\*1e+09^2\*1e-27

                   2.2841e-26\*1e+09^2/6.24150934e+18/1e-27'

                   # kappa\_c\*eV\_J\*nm\_m^2\*d

                   # M\*nm\_m^2/eV\_J/d

  [../]

* # Define constant values kappa\_c and M.
* # d is a scaling factor that makes it easier for the solution to converge without changing the results. It is defined in each of the materials and must have the same value in each one.

### S2\_testmodel.i ([s2\_fasttest.i](https://github.com/idaholab/moose/blob/devel/modules/phase_field/tutorials/spinodal_decomposition/s2_fasttest.i))

## A phase field model of pressure-assisted sintering (Dzepina et al., 2019)

There is an issue with the equation for Allen mobility below eq. (9) in this paper.



where the variables are defined in Figure 17. The dimension of must be but it is not.

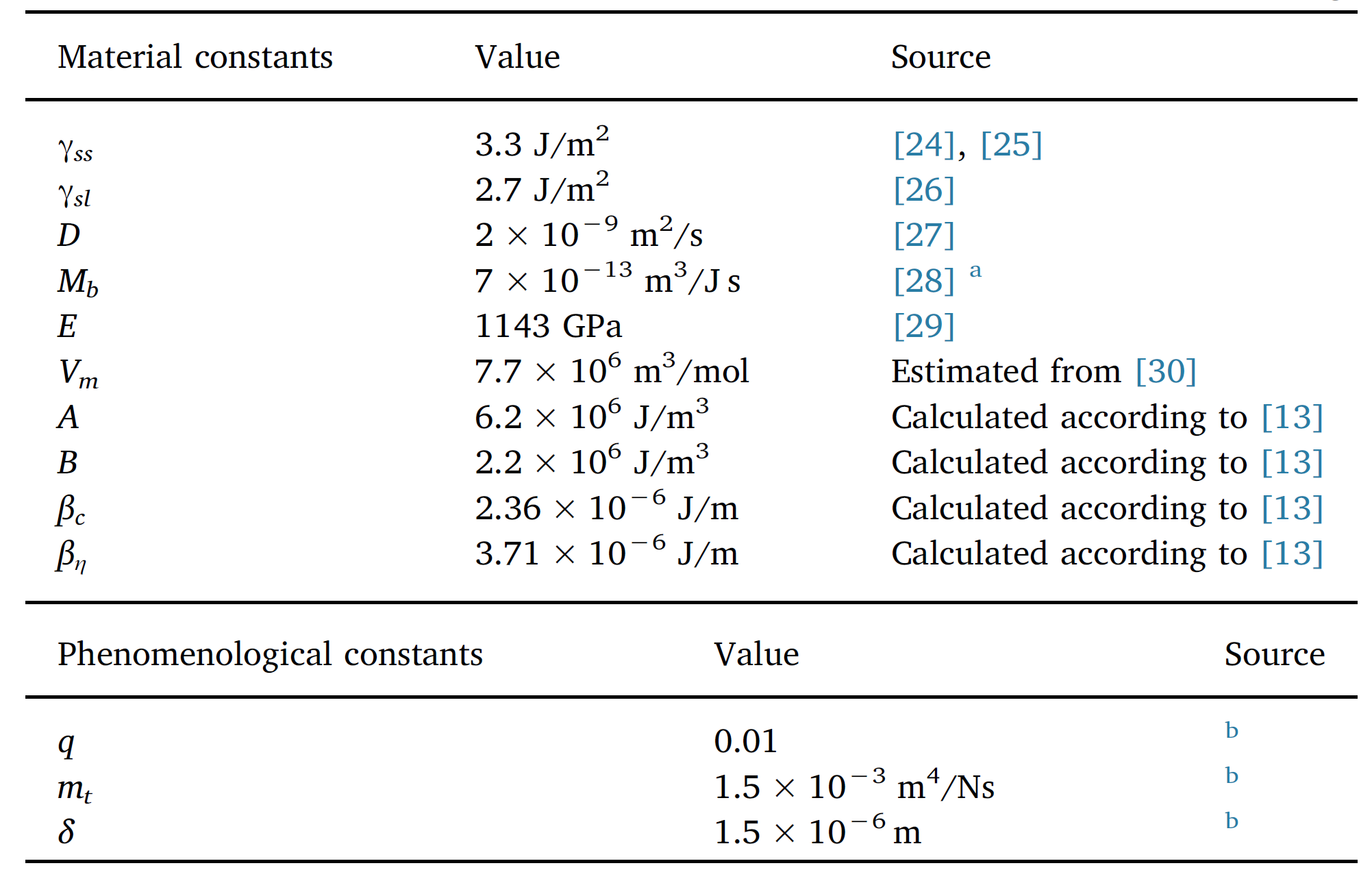


Figure 18 Table of material constants and model parameters for diamond-nickel sintering.

# Thermomechanical coupling

## Simple examples

### circle\_thermal\_expansion\_stress.i (combined examples)

This example problem demonstrates coupling heat conduction with mechanics. A circular domain has as uniform heat source that increases with time and a fixed temperature on the outer boundary, resulting in a temperature gradient. This results in heterogeneous thermal expansion, where it is pinned in the center. Looking at the hoop stress demonstrates why fuel pellets have radial cracks that extend from the outer boundary to about halfway through the radius. The problem is run with length units of microns.

Figure 19 Mesh and results of simulation of the heat conduction in a circular domain.

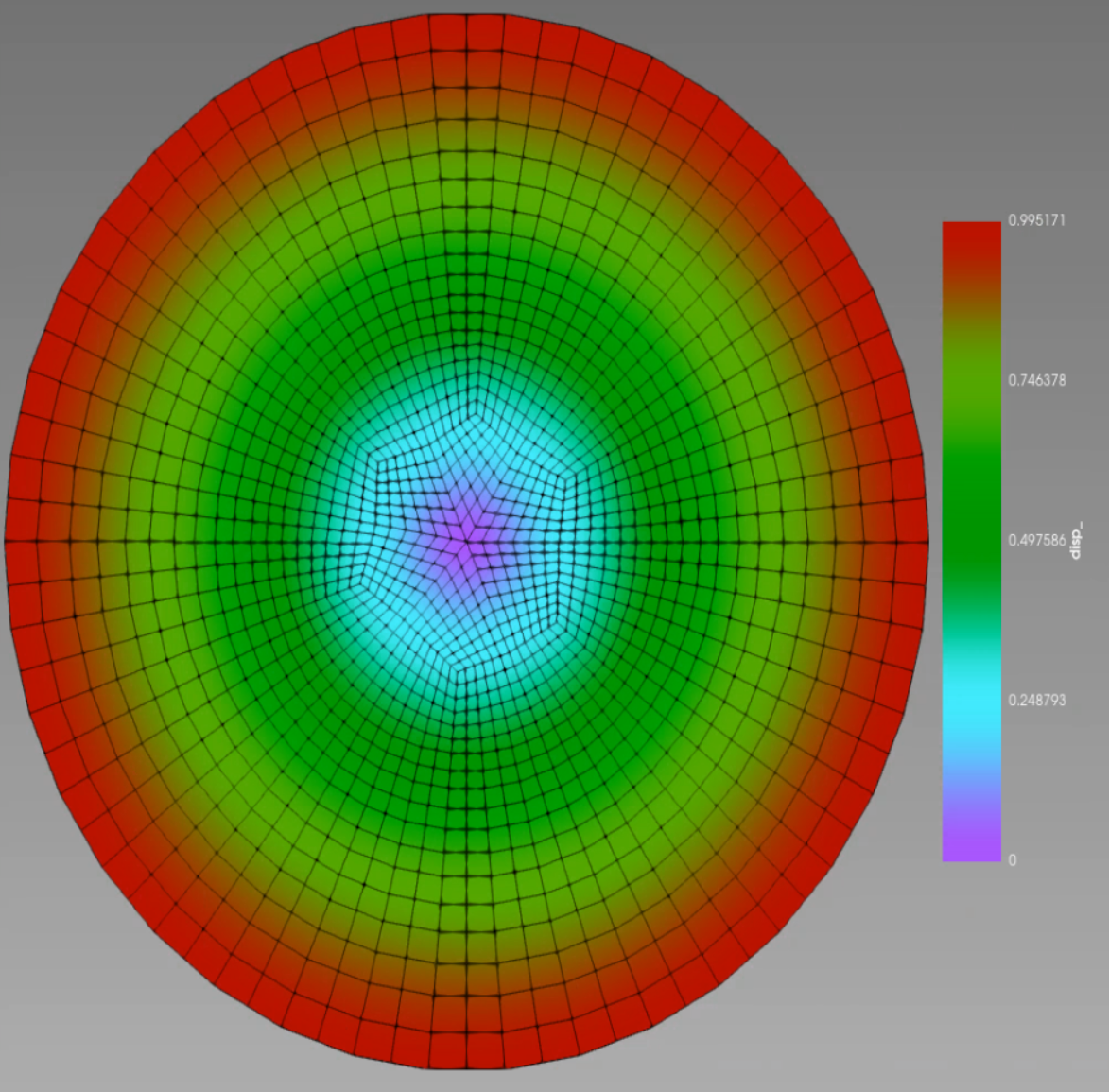


Table Script for thermomechanical coupling in a circular domain.

|  |  |  |
| --- | --- | --- |
| [Mesh]    #Circle mesh has a radius of 1000 units    type = *FileMesh*    file = circle.e    uniform\_refine = 1  []  [Variables]    # We solve for the temperature and the displacements    [./T]      initial\_condition = 800      scaling = 1e7    [../]    [./disp\_x]    [../]    [./disp\_y]    [../]  []  [AuxVariables]    [./radial\_stress]      order = CONSTANT      family = MONOMIAL    [../]    [./hoop\_stress]      order = CONSTANT      family = MONOMIAL    [../]  []  [Kernels]    active = 'TensorMechanics htcond Q\_function'    [./htcond] #Heat conduction equation      type = *HeatConduction*      variable = T    [../]    [./TensorMechanics] #Action that creates equations for disp\_x and disp\_y      displacements = 'disp\_x disp\_y'    [../]    [./Q\_function] #Heat generation term      type = *BodyForce*      variable = T      value = 1      function = 0.8e-9\*t    [../]  [] | [AuxKernels]    [./radial\_stress] #Calculates radial stress from cartesian      type = *CylindricalRankTwoAux*      variable = radial\_stress      rank\_two\_tensor = stress      index\_j = 0      index\_i = 0      center\_point = '0 0 0'    [../]    [./hoop\_stress] #Calculates hoop stress from cartesian      type = *CylindricalRankTwoAux*      variable = hoop\_stress      rank\_two\_tensor = stress      index\_j = 1      index\_i = 1      center\_point = '0 0 0'    [../]  []  [BCs]    [./outer\_T] #Temperature on outer edge is fixed at 800K      type = *DirichletBC*      variable = T      boundary = 1      value = 800    [../]    [./outer\_x] #Displacements in the x-direction are fixed in the center      type = *DirichletBC*      variable = disp\_x      boundary = 2      value = 0    [../]    [./outer\_y] #Displacements in the y-direction are fixed in the center      type = *DirichletBC*      variable = disp\_y      boundary = 2      value = 0    [../]  []  [Materials]    [./thcond] #Thermal conductivity is set to 5 W/mK      type = *GenericConstantMaterial*      block = 1      prop\_names = 'thermal\_conductivity'      prop\_values = '5e-6'    [../] | [./iso\_C] #Sets isotropic elastic constants      type = *ComputeElasticityTensor*      fill\_method = symmetric\_isotropic      C\_ijkl = '2.15e5 0.74e5'      block = 1    [../]    [./strain] #We use small deformation mechanics      type = *ComputeSmallStrain*      displacements = 'disp\_x disp\_y'      block = 1      eigenstrain\_names = eigenstrain    [../]    [./stress] #We use linear elasticity      type = *ComputeLinearElasticStress*      block = 1    [../]    [./thermal\_strain]      type= *ComputeThermalExpansionEigenstrain*      thermal\_expansion\_coeff = 1e-6      temperature = T      stress\_free\_temperature = 273      block = 1      eigenstrain\_name = eigenstrain    [../]  []  [Executioner]    type = *Transient*    scheme = bdf2    num\_steps = 10    solve\_type = PJFNK    petsc\_options\_iname = '-pc\_type -pc\_hypre\_type -ksp\_gmres\_restart'    petsc\_options\_value = 'hypre boomeramg 101'    l\_max\_its = 30    nl\_max\_its = 10    nl\_abs\_tol = 1e-9    l\_tol = 1e-04  []  [Outputs]    exodus = true    perf\_graph = true  [] |

The script analysis.

* Read mesh from file circle.e
* Define variables T, disp\_x, disp\_y.
* Define initial conditions for T on the fly. Note “scaling = 1e7”[[1]](#footnote-1)
* Additional variables are calculated in AuxKernels
* Kernels: htconduction, TensorMechanics,Q\_function[[2]](#footnote-2)

## Darcy Flow Tutorial

### Equations of motion (<https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Problem/>)



where   is the fluid velocity, is porosity, is the permeability tensor, is fluid viscosity, is the pressure,  is the gravity vector, and is the temperature. The parameters , , and are the porosity-dependent density, heat capacity, and thermal conductivity of the combined fluid/solid medium, defined by:



where is the porosity, is the specific heat, and the subscripts and refer to fluid and solid, respectively.

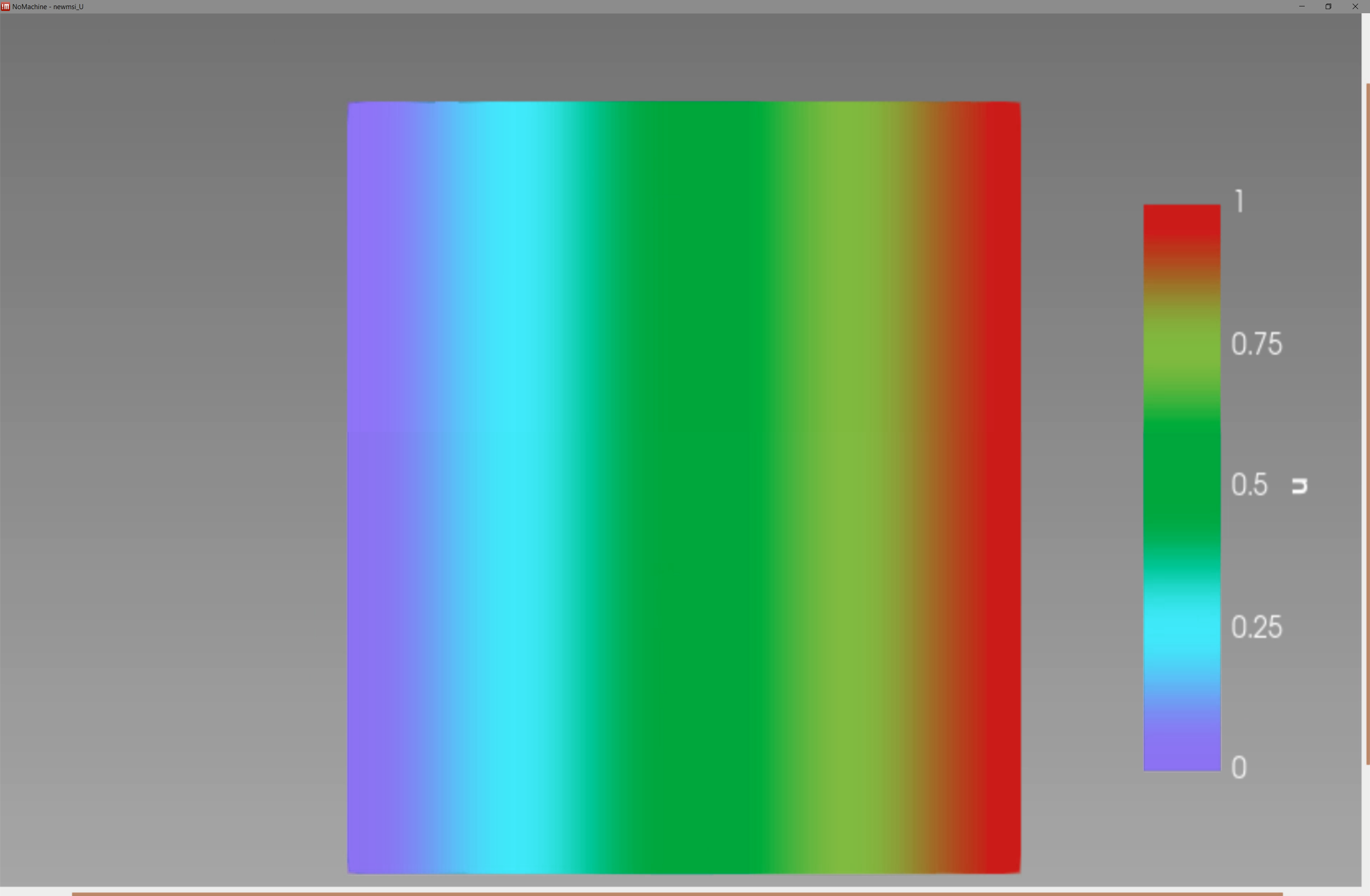
### Steps

#### Step 01 (<https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Step01/>)

Solve



Figure 20 Solution of Step1.i



To use step\*.py files for plotting add

export PYTHONPATH=/full/path/to/moose/python:$PYTHONPATH

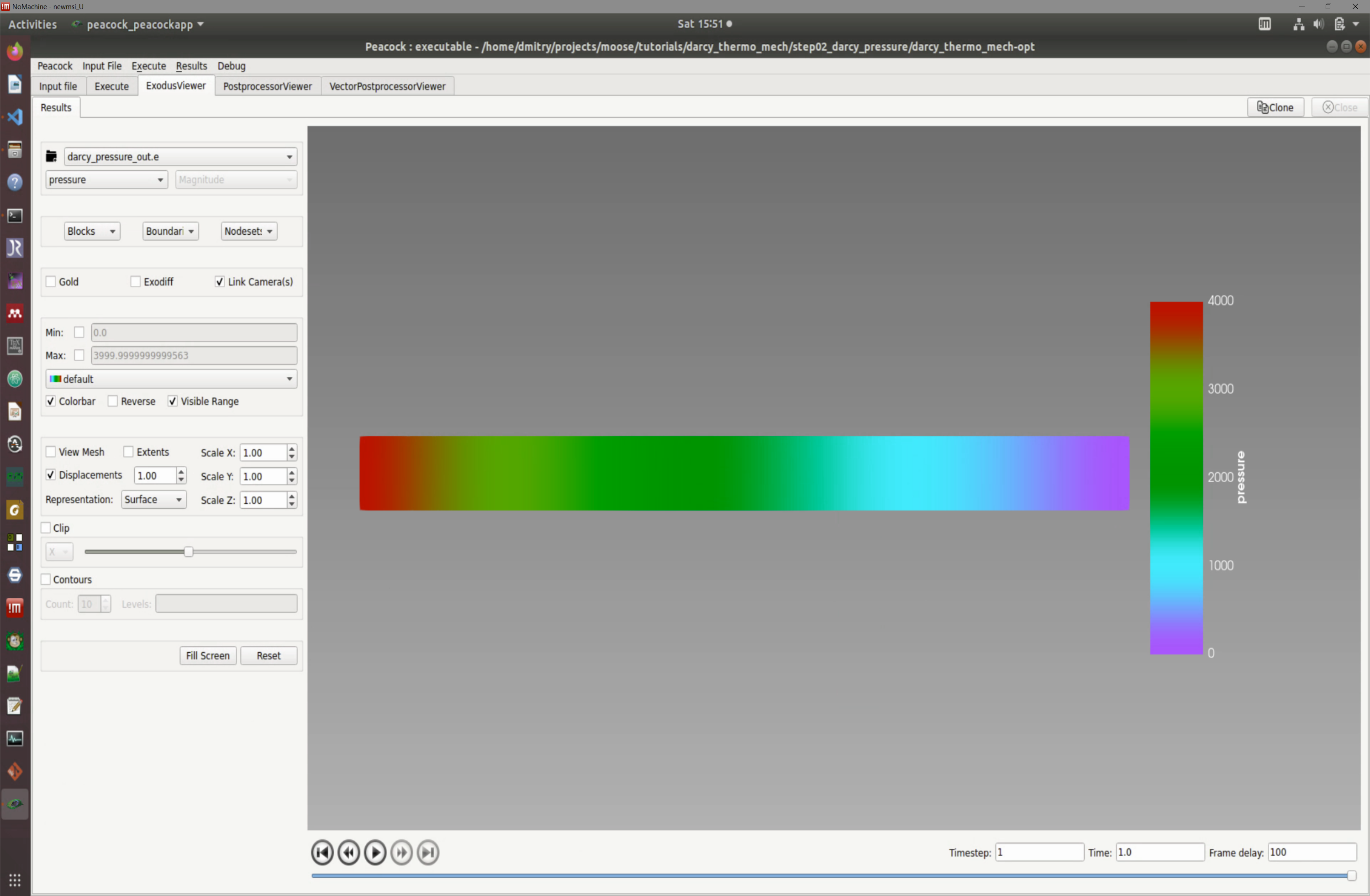
to .bashrc file

#### Step 2 (<https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Step02/> )



Figure 21 Solution of Step2.i

#### Step 3 (<https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Step03/> )



Note that so far only minor changes were introduced to the script in Table 1.

1. The kernel was changed from

  [diffusion]

    type = *ADDiffusion*  # Laplacian operator using automatic differentiation

    variable = pressure # Operate on the "pressure" variable from above

  []

to

  [darcy\_pressure]

    type = *DarcyPressure*

    variable = pressure

  []

1. Material section was added

[Materials]

  [column]

    type = *PackedColumn*

  []

[

1. Solver\_type was changed from NEWTON to PJFNK [[3]](#footnote-3)

#### Step 4

Added AuxVariables and Kernels for velocity

Table Additions at step 4

|  |  |
| --- | --- |
| [AuxVariables]    [velocity\_x]      order = CONSTANT      family = MONOMIAL    []    [velocity\_y]      order = CONSTANT      family = MONOMIAL    []    [velocity\_z]      order = CONSTANT      family = MONOMIAL    []    [velocity]      order = CONSTANT      family = MONOMIAL\_VEC    []  []  [AuxKernels]    [velocity]      type = *DarcyVelocity*      variable = velocity      execute\_on = timestep\_end      pressure = pressure    [] | [velocity\_x]      type = *VectorVariableComponentAux*      variable = velocity\_x      component = x      execute\_on = timestep\_end      vector\_variable = velocity    []    [velocity\_y]      type = *VectorVariableComponentAux*      variable = velocity\_y      component = y      execute\_on = timestep\_end      vector\_variable = velocity    []    [velocity\_z]      type = *VectorVariableComponentAux*      variable = velocity\_z      component = z      execute\_on = timestep\_end      vector\_variable = velocity    []  [] |

#### Step 5. Heat conduction



Table Heat conduction: step5c\_outflow.i (check the steady version Step5a\_steady.i)

|  |  |
| --- | --- |
| [Mesh]    type = *GeneratedMesh*    dim = 2    nx = 100    ny = 10    xmax = 0.304 # Length of test chamber    ymax = 0.0257 # Test chamber radius  []  [Variables]    [temperature]      initial\_condition = 300 # Start at room temperature    []  []  [Kernels]    [heat\_conduction]      type = *ADHeatConduction*      variable = temperature    []    [heat\_conduction\_time\_derivative]      type = *ADHeatConductionTimeDerivative*      variable = temperature    []  []  [BCs]    [inlet\_temperature]      type = *DirichletBC*      variable = temperature      boundary = left      value = 350 # (K)    [] | [outlet\_temperature]      type = *HeatConductionOutflow*      variable = temperature      boundary = right    []  []  [Materials]    [steel]      type = *GenericConstantMaterial*      prop\_names = 'thermal\_conductivity specific\_heat density'      prop\_values = '18 466 8000' # W/m\*K, J/kg-K, kg/m^3 @ 296K    []  []  [Problem]    type = *FEProblem*    coord\_type = RZ    rz\_coord\_axis = X  []  [Executioner]    type = *Transient*    num\_steps = 100    solve\_type = NEWTON    petsc\_options\_iname = '-pc\_type -pc\_hypre\_type'    petsc\_options\_value = 'hypre boomeramg'  []  [Outputs]    exodus = true  [] |

#### Step 6 Darcy and heat conduction coupling

Table Step6\_coupled.i

|  |  |  |
| --- | --- | --- |
| [Mesh]    type = *GeneratedMesh*    dim = 2    nx = 200    ny = 10    xmax = 0.304 # Length of test chamber    ymax = 0.0257 # Test chamber radius  []  [Variables]    [pressure]    []    [temperature]      initial\_condition = 300 # Start at room temperature    []  []  [AuxVariables]    [velocity]      order = CONSTANT      family = MONOMIAL\_VEC    []  []  [Kernels]    [darcy\_pressure]      type = *DarcyPressure*      variable = pressure    []    [heat\_conduction]      type = *ADHeatConduction*      variable = temperature    []    [heat\_conduction\_time\_derivative]      type = *ADHeatConductionTimeDerivative*      variable = temperature    []    [heat\_convection]      type = *DarcyAdvection*      variable = temperature      pressure = pressure    []  [] | [AuxKernels]    [velocity]      type = *DarcyVelocity*      variable = velocity      execute\_on = timestep\_end      pressure = pressure    []  []    [BCs]    [inlet]      type = *DirichletBC*      variable = pressure      boundary = left      value = 4000 # (Pa) From Figure 2 from paper.  First data point for 1mm spheres.    []    [outlet]      type = *DirichletBC*      variable = pressure      boundary = right      value = 0 # (Pa) Gives the correct pressure drop from Figure 2 for 1mm spheres    []    [inlet\_temperature]      type = *FunctionDirichletBC*      variable = temperature      boundary = left      function = 'if(t<0,350+50\*t,350)'    []    [outlet\_temperature]      type = *HeatConductionOutflow*      variable = temperature      boundary = right    []  [] | [Materials]    [column]      type = *PackedColumn*      temperature = temperature      radius = 1    []  []  [Problem]    type = *FEProblem*    coord\_type = RZ    rz\_coord\_axis = X  []  [Executioner]    type = *Transient*    solve\_type = NEWTON    automatic\_scaling = true    petsc\_options\_iname = '-pc\_type -pc\_hypre\_type'    petsc\_options\_value = 'hypre boomeramg'    end\_time = 100    dt = 0.25    start\_time = -1    steady\_state\_tolerance = 1e-5    steady\_state\_detection = true    [TimeStepper]      type = *FunctionDT*      function = 'if(t<0,0.1,0.25)'    []  []  [Outputs]    exodus = true  [] |

It can be seen form the Table 4 that the coupled set of equations is a straightforward combination of the individual solvers considered earlier. Similarly, one can add Mechanics module to the calculations.

One has to remember though that “Mechanics” Mechanics is a complicated beast, see <https://www.mooseframework.org/modules/tensor_mechanics/index.html> . The steps required to include “Mechanics” module are discussed here

<https://mooseframework.inl.gov/old/wiki/MooseTutorials/DarcyThermoMechanical/Step09/> .

#### Step9

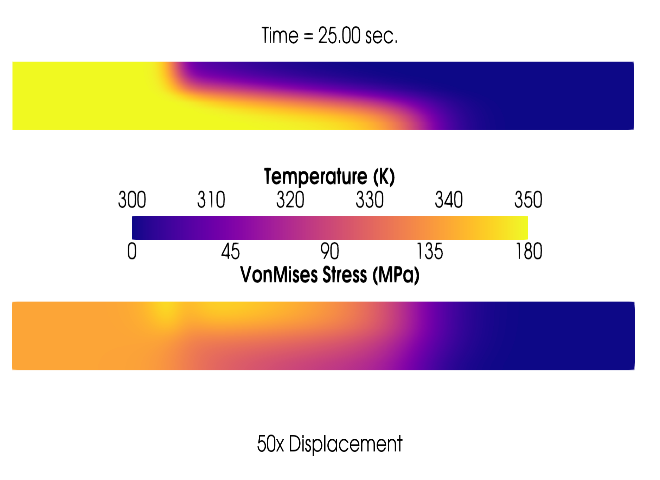


Table Step9.i

|  |  |  |  |
| --- | --- | --- | --- |
| [GlobalParams]    displacements = 'disp\_r disp\_z'  []  [Mesh]    type = *GeneratedMesh*    dim = 2    ny = 200    nx = 10    ymax = 0.304 # Length of test chamber    xmax = 0.0257 # Test chamber radius  []  [MeshModifiers]    [bottom]      type = *SubdomainBoundingBox*      location = inside      bottom\_left = '0 0 0'      top\_right = '0.01285 0.304 0'      block\_id = 1    []  []  [Variables]    [pressure]    []    [temperature]      initial\_condition = 300 # Start at room temperature    []  []  [AuxVariables]    [velocity]      order = CONSTANT      family = MONOMIAL\_VEC    []  []  [Modules/TensorMechanics/Master]    [all]      # This block adds all of the proper Kernels, strain calculators, and Variables      # for TensorMechanics in the correct coordinate system (autodetected)      add\_variables = true      strain = FINITE      eigenstrain\_names = eigenstrain      use\_automatic\_differentiation = true      generate\_output = 'vonmises\_stress elastic\_strain\_xx elastic\_strain\_yy strain\_xx strain\_yy'    []  [] | [Kernels]    [darcy\_pressure]      type = *DarcyPressure*      variable = pressure    []    [heat\_conduction]      type = *ADHeatConduction*      variable = temperature    []    [heat\_conduction\_time\_derivative]      type = *ADHeatConductionTimeDerivative*      variable = temperature    []    [heat\_convection]      type = *DarcyAdvection*      variable = temperature      pressure = pressure    []  []  [AuxKernels]    [velocity]      type = *DarcyVelocity*      variable = velocity      execute\_on = timestep\_end      pressure = pressure    []  []  [BCs]    [inlet]      type = *DirichletBC*      variable = pressure      boundary = bottom      value = 4000 # (Pa) From Figure 2 from paper.  First data point for 1mm spheres.    []    [outlet]      type = *DirichletBC*      variable = pressure      boundary = top      value = 0 # (Pa) Gives the correct pressure drop from Figure 2 for 1mm spheres    []    [inlet\_temperature]      type = *FunctionDirichletBC*      variable = temperature      boundary = bottom      function = 'if(t<0,350+50\*t,350)'    []    [outlet\_temperature]      type = *HeatConductionOutflow*      variable = temperature      boundary = top    []    [] | [hold\_inlet]      type = *DirichletBC*      variable = disp\_z      boundary = bottom      value = 0    []    [hold\_center]      type = *DirichletBC*      variable = disp\_r      boundary = left      value = 0    []    [hold\_outside]      type = *DirichletBC*      variable = disp\_r      boundary = right      value = 0    []  []  [Materials]    viscosity\_file = data/water\_viscosity.csv    density\_file = data/water\_density.csv    thermal\_conductivity\_file = data/water\_thermal\_conductivity.csv    specific\_heat\_file = data/water\_specific\_heat.csv    thermal\_expansion\_file = data/water\_thermal\_expansion.csv    [column\_top]      type = *PackedColumn*      block = 0      temperature = temperature      radius = 1.15      fluid\_viscosity\_file = ${viscosity\_file}      fluid\_density\_file = ${density\_file}      fluid\_thermal\_conductivity\_file = ${thermal\_conductivity\_file}      fluid\_specific\_heat\_file = ${specific\_heat\_file}      fluid\_thermal\_expansion\_file = ${thermal\_expansion\_file}    []    [column\_bottom]      type = *PackedColumn*      block = 1      temperature = temperature      radius = 1      fluid\_viscosity\_file = ${viscosity\_file}      fluid\_density\_file = ${density\_file}      fluid\_thermal\_conductivity\_file = ${thermal\_conductivity\_file}      fluid\_specific\_heat\_file = ${specific\_heat\_file}      fluid\_thermal\_expansion\_file = ${thermal\_expansion\_file}    [] | [elasticity\_tensor]      type = *ComputeIsotropicElasticityTensor*      youngs\_modulus = 200e9 # (Pa) from wikipedia      poissons\_ratio = .3 # from wikipedia    []    [elastic\_stress]      type = *ADComputeFiniteStrainElasticStress*    []    [thermal\_strain]      type = *ADComputeThermalExpansionEigenstrain*      stress\_free\_temperature = 300      eigenstrain\_name = eigenstrain      temperature = temperature      thermal\_expansion\_coeff = 1e-5 # TM modules doesn't support material property, but it will    []  []  [Postprocessors]    [average\_temperature]      type = *ElementAverageValue*      variable = temperature    []  []  [Problem]    type = *FEProblem*    coord\_type = RZ  []  [Executioner]    type = *Transient*    start\_time = -1    end\_time = 200    steady\_state\_tolerance = 1e-7    steady\_state\_detection = true    dt = 0.25    solve\_type = PJFNK    automatic\_scaling = true    compute\_scaling\_once = false    petsc\_options\_iname = '-pc\_type'    petsc\_options\_value = 'lu'    #petsc\_options\_iname = '-pc\_type -pc\_hypre\_type -ksp\_gmres\_restart'    #petsc\_options\_value = 'hypre boomeramg 500'    line\_search = none    [TimeStepper]      type = *FunctionDT*      function = 'if(t<0,0.1,0.25)'    []  []  [Outputs]    [out]      type = *Exodus*      elemental\_as\_nodal = true    [] |

Figure 22 Results of simulatins at Step9.i

# Using SINK boundary conditions



Question: Is the parsed function of MOOSE capable of moving a heat source (e.g. laser scanning) ? (<https://groups.google.com/forum/#!topic/moose-users/J5M6-MNPdiI> )

Answer: Sure - if it's an actual boundary condition (like, it's applied on the boundary) then it's pretty straightforward to do with a ***parsed function*** and ***FunctionBC***.

## Basic sink formulation

The basic sink is

where is a MOOSE Function of time and position on the boundary.

If then the boundary condition will act as a sink, while if the boundary condition acts as a source. If applied to a fluid-component equation, the function has units kg.m−2.s−1. If applied to the heat equation, the function has units J.m−2.s−1. The units of are potentially modified if the extra building blocks enumerated below are used (but the units of the final result, , will always be kg.m−2.s−1 or J.m−2.s−1).

If the fluid flow through the boundary is required, use the ***save\_in*** command which will save the sink strength to an ***AuxVariable***. This AuxVariable will be the flux (kg.s−1 or J.s−1) from each node on the boundary, which is the product of and the area attributed to that node. If the total flux (kg.s−1 or J.s−1) through the boundary is required, integrate the AuxVariable over the boundary using a ***NodalSum*** Postprocessor.

This basic sink boundary condition is implemented in ***PorousFlowSink*** (<https://www.mooseframework.org/source/bcs/PorousFlowSink.html> ).

## Sink tests ([https://www.mooseframework.org/modules/porous\_flow/tests/sinks/sinks\_tests.html#](https://www.mooseframework.org/modules/porous_flow/tests/sinks/sinks_tests.html)! )

### Test 1

Table . Script for Test1 of sink

|  |  |  |
| --- | --- | --- |
| # apply a sink flux and observe the correct behavior  [Mesh]    type = *GeneratedMesh*    dim = 3    nx = 1    ny = 1    nz = 1    xmin = 0    xmax = 1    ymin = 0    ymax = 1    zmin = 0    zmax = 2  []  [GlobalParams]    PorousFlowDictator = dictator  []  [UserObjects]    [./dictator]      type = *PorousFlowDictator*      porous\_flow\_vars = 'pp'      number\_fluid\_phases = 1      number\_fluid\_components = 1    [../]    [./pc]      type = *PorousFlowCapillaryPressureVG*      m = 0.5      alpha = 1    [../]  []  [Variables]    [./pp]    [../]  []  [ICs]    [./pp]      type = *FunctionIC*      variable = pp      function = y+1    [../]  []  [Kernels]    [./mass0]      type = *PorousFlowMassTimeDerivative*      fluid\_component = 0      variable = pp    [../]  []  [Modules]    [./FluidProperties]      [./simple\_fluid]        type = *SimpleFluidProperties*        bulk\_modulus = 1.3        density0 = 1.1        thermal\_expansion = 0        viscosity = 1.1      [../]    [../]  []  [Materials]    [./temperature]      type = *PorousFlowTemperature*    [../]    [./ppss]      type = *PorousFlow1PhaseP*      porepressure = pp      capillary\_pressure = pc    [../]    [./massfrac]      type = *PorousFlowMassFraction*    [../] | [./simple\_fluid]      type = *PorousFlowSingleComponentFluid*      fp = simple\_fluid      phase = 0    [../]    [./porosity]      type = *PorousFlowPorosityConst*      porosity = 0.1    [../]    [./permeability]      type = *PorousFlowPermeabilityConst*      permeability = '1E-5 0 0 0 1E-5 0 0 0 1E-5'    [../]    [./relperm]      type = *PorousFlowRelativePermeabilityCorey*      n = 2      phase = 0    [../]  []  [AuxVariables]    [./flux\_out]    [../]    [./xval]    [../]    [./yval]    [../]  []  [ICs]    [./xval]      type = *FunctionIC*      variable = xval      function = x    [../]    [./yval]      type = *FunctionIC*      variable = yval      function = y    [../]  []  [Functions]    [./mass00]      type = *ParsedFunction*      value = 'vol\*por\*dens0\*exp(pp/bulk)'      vars = 'vol por dens0 pp bulk'      vals = '0.25 0.1 1.1 p00 1.3'    [../]    [./mass01]      type = *ParsedFunction*      value = 'vol\*por\*dens0\*exp(pp/bulk)'      vars = 'vol por dens0 pp bulk'      vals = '0.25 0.1 1.1 p01 1.3'    [../]    [./expected\_mass\_change00]      type = *ParsedFunction*      value = 'fcn\*perm\*dens0\*exp(pp/bulk)/visc\*area\*dt'      vars = 'fcn perm dens0 pp bulk visc area dt'      vals = '6   1    1      0  1.3  1  0.5  1E-3'    [../]  []  [Postprocessors]    [./p00]      type = *PointValue*      point = '0 0 0'      variable = pp      execute\_on = 'initial timestep\_end'    [../]    [./m00]      type = *FunctionValuePostprocessor*      function = mass00      execute\_on = 'initial timestep\_end'    [../] | [./del\_m00]      type = *FunctionValuePostprocessor*      function = expected\_mass\_change00      execute\_on = 'timestep\_end'    [../]     [./p10]      type = *PointValue*      point = '1 0 0'      variable = pp      execute\_on = 'initial timestep\_end'    [../]   [./p01]      type = *PointValue*      point = '0 1 0'      variable = pp      execute\_on = 'initial timestep\_end'    [../]    [./m01]      type = *FunctionValuePostprocessor*      function = mass01      execute\_on = 'initial timestep\_end'    [../]    [./p11]      type = *PointValue*      point = '1 1 0'      variable = pp      execute\_on = 'initial timestep\_end'    [../]  []  [BCs]    [./flux]      type = *PorousFlowSink*      boundary = 'left'      variable = pp      use\_mobility = false      use\_relperm = true      fluid\_phase = 0      flux\_function = 6      save\_in = flux\_out    [../]  []  [Preconditioning]    [./andy]      type = *SMP*      full = true      petsc\_options\_iname = '-ksp\_type -pc\_type -sub\_pc\_type -snes\_max\_it -sub\_pc\_factor\_shift\_type -pc\_asm\_overlap'      petsc\_options\_value = 'gmres asm lu 10000 NONZERO 2'    [../]  []  [Executioner]    type = *Transient*    solve\_type = Newton    dt = 1E-3    end\_time = 1E-2    nl\_rel\_tol = 1E-12    nl\_abs\_tol = 1E-12  []  [Outputs]    file\_base = s01    [./console]      type = *Console*      execute\_on = 'nonlinear linear'    [../]    [./csv]      type = *CSV*      execute\_on = 'initial timestep\_end'    [../]  [] |

### Gaussian moving heat source

Table Moving Gaussian source for pure heat conduction in generic material

|  |  |  |
| --- | --- | --- |
| # This test solves a 1D transient heat equation  # The error is caclulated by comparing to the analytical solution  # The problem setup and analytical solution are taken from "Advanced Engineering  # Mathematics, 10th edition" by Erwin Kreyszig.  # http://www.amazon.com/Advanced-Engineering-Mathematics-Erwin-Kreyszig/dp/0470458364  # It is Example 1 in section 12.6 on page 561  [Mesh]    type = *GeneratedMesh*    dim = 3    nx = 30    ny = 30    nz = 30    xmax = 30    ymax = 30    zmax = 10  []  [Variables]    [./T]    [../]  []  [Kernels]    [./HeatSource]      type = *HeatSource*      # function = '1\*sin(3.14159\*x/20)\*sin(3.14159\*y/80)\*sin(3.14159\*z/20)'      function = '10\*exp(-((x-(30-2\*t))^2+(y-(30-2\*t))^2)/20)\*exp(-z/1.5)'      variable = T    [../]    [./HeatDiff]      type = *HeatConduction*      variable = T    [../]    [./HeatTdot]      type = *HeatConductionTimeDerivative*      variable = T    [../]  [] | [BCs]    [./left]      type = *DirichletBC*      variable = T      boundary = left      value = 0    [../]    [./right]      type = *DirichletBC*      variable = T      boundary = right      value = 0    [../]    [./top]      type = *DirichletBC*      variable = T      boundary = top      value = 0    [../]    [./bottom]      type = *DirichletBC*      variable = T      boundary = bottom      value = 0    [../]  []  [Materials]    [./k]      type = *GenericConstantMaterial*      prop\_names = 'thermal\_conductivity'      prop\_values = '0.95' #copper in cal/(cm sec C)      block = 0    [../]    [./cp]      type = *GenericConstantMaterial*      prop\_names = 'specific\_heat'      prop\_values = '0.092' #copper in cal/(g C)      block = 0    [../] | [./rho]      type = *GenericConstantMaterial*      prop\_names = 'density'      prop\_values = '8.92' #copper in g/(cm^3)      block = 0    [../]  []  [Postprocessors]    [./error]      type = *NodalL2Error*      function = '100\*sin(3.14159\*x/20)\*exp(-0.95/(0.092\*8.92)\*3.14159^2/20^2\*t)' #T(x,t) = 100sin(pix/L)exp(-rho/(cp k) pi^2/L^2 t)      variable = T    [../]  []  [Executioner]    type = *Transient*    scheme = bdf2    nl\_rel\_tol = 1e-12    l\_tol = 1e-8    dt = 0.025    #end\_time = 100    end\_time = 10    petsc\_options\_iname = '-pc\_type -pc\_hypre\_type'    petsc\_options\_value = 'hypre boomeramg'  []  [Outputs]    exodus = true    print\_perf\_log = true  [] |

# Adding heat source to the phase-field model of sintering

We review the following publications [7] and [10].

In the paper by X. Zhang and Y. Liao [10] the authors develop MOOSE model of Selective Laser Sintering by adding thermal module to the Biswas [1], [2] model of sintering



# Phase-field tutorial (**Spinodal Decomposition of Iron-Chromium Alloy)**









# Add temperature dependent diffusion coefficients.

## Step 1: Make a Simple Test Model

We would like to modify material section to include variable diffusion coefficients (coded using C-function in Sudipta’s project) directly into the script.

Table (left) Material properties coded directly in the script and (right) using C-code PFDiffusionGrowth.C

|  |  |
| --- | --- |
| [Materials]    # d is a scaling factor that makes it easier for the solution to converge    # without changing the results. It is defined in each of the materials and    # must have the same value in each one.    [./constants]      # Define constant values kappa\_c and M. Eventually M will be replaced with      # an equation rather than a constant.      type = *GenericFunctionMaterial*      prop\_names = 'kappa\_c M'      prop\_values = '8.125e-16\*6.24150934e+18\*1e+09^2\*1e-27                     2.2841e-26\*1e+09^2/6.24150934e+18/1e-27'                     # kappa\_c\*eV\_J\*nm\_m^2\*d                     # M\*nm\_m^2/eV\_J/d    [../]    [./local\_energy]      # Defines the function for the local free energy density as given in the      # problem, then converts units and adds scaling factor.      type = *DerivativeParsedMaterial*      f\_name = f\_loc      args = c      constant\_names = 'A   B   C   D   E   F   G  eV\_J  d'      constant\_expressions = '-2.446831e+04 -2.827533e+04 4.167994e+03 7.052907e+03                              1.208993e+04 2.568625e+03 -2.354293e+03                              6.24150934e+18 1e-27'      function = 'eV\_J\*d\*(A\*c+B\*(1-c)+C\*c\*log(c)+D\*(1-c)\*log(1-c)+                  E\*c\*(1-c)+F\*c\*(1-c)\*(2\*c-1)+G\*c\*(1-c)\*(2\*c-1)^2)'    [../]  [] | [Materials]    [./chemical\_free\_energy]      type = *SinteringFreeEnergy*      block = 0      c = c      v = 'gr0 gr1 gr2 gr3 gr4 gr5 gr6 gr7 gr8'      f\_name = Fc      derivative\_order = 2      #outputs = console    [../]    [./CH\_mat]      type = *PFDiffusionGrowth*      block = 0      rho = c      v = 'gr0 gr1 gr2 gr3 gr4 gr5 gr6 gr7 gr8'      outputs = console    [../]    [./constant\_mat]      type = *GenericConstantMaterial*      block = 0      prop\_names = '  A    B   L   kappa\_op kappa\_c'      prop\_values = '16.0 1.0 1.0  0.5      1.0    '    [../]  [] |

There are at least two ways to introduce temperature dependent diffusion coefficients.

The first one is shown in the Table 8 (left) and it includes required functions (including definition of the units) directly into the script. 

The 2nd one requires C++ coding in MOOSE. Here we could try to modify

With units of the units of are and the units of are , and are gradient energy coefficients with the units .



Mobility coefficients () are estimated from corresponding diffusion coefficients as

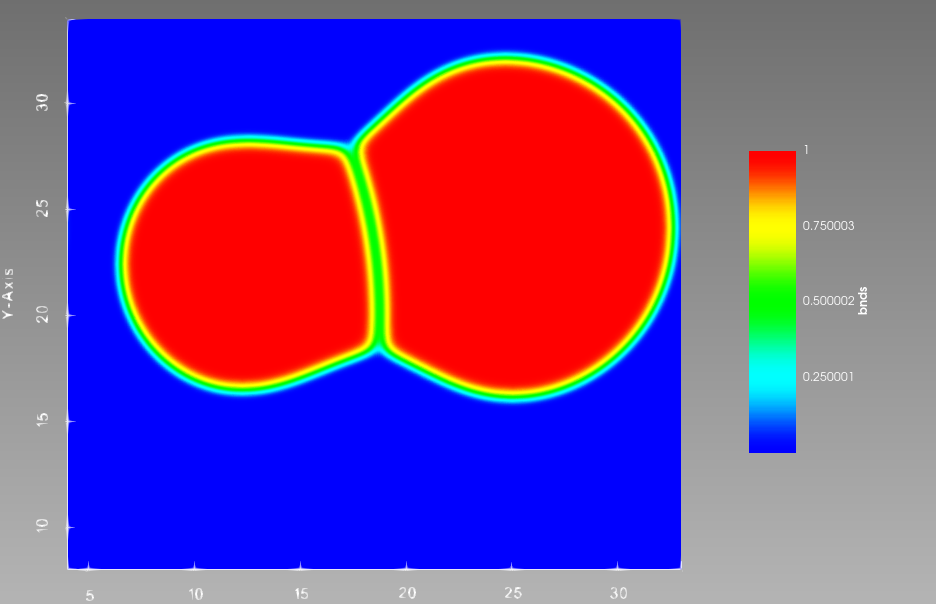
and are the surface and the grain boundary energy, is the interface width or the grain boundary thickness. are normalization constants.

Consider one example: , , , .

Even simper example: while dimensionless value is ~1.

Figure Reduced system with two particles.

We first try scripting approach. First, we reduce the number of particles to two p0articles.



Next, we follow the example polycrystal\_action.i see Table 9.

*Table 9 Example DerivativeParsedMaterial*

|  |  |
| --- | --- |
| [./chemical\_free\_energy]      type = *DerivativeParsedMaterial*      f\_name = F      args = 'c eta0 eta1'      constant\_names = 'barr\_height  cv\_eq'      constant\_expressions = '0.1          1.0e-2'      function = 16\*barr\_height\*(c-cv\_eq)^2\*(1-cv\_eq-c)^2+eta0\*(1-eta0)\*c+eta1\*(1-eta1)\*c      derivative\_order = 2    [../] | [./chemical\_free\_energy]      type = *DerivativeParsedMaterial*      f\_name = Fc      args = 'c gr0 gr1'      constant\_names = 'A  B'      constant\_expressions = '16.0 1.0'      function = A\*c^2\*(1-c)^2+B\*(c^2+6\*(1-c)\*(gr0^2+gr1^2)-4\*(2 - c) \* (gr0^3 + gr1^3) + 3\*(gr0^2+gr1^2)^2)      derivative\_order = 2    [../] |

The substitution of the c-code for chemical free energy with MOOSE script seems to be working OK. Our next step is to substitute PFDiffusionGrowth.c code with MOOSE script, see Table 10.

Table 10 Original Material Functions in Sudipta’s code

|  |  |
| --- | --- |
| [./CH\_mat]      type = *PFDiffusionGrowth*      block = 0      rho = c      v = 'gr0 gr1 gr2 gr3 gr4 gr5 gr6 gr7 gr8'      outputs = console    [../]    [./constant\_mat]      type = *GenericConstantMaterial*      block = 0      prop\_names = '  A    B   L    kappa\_op kappa\_c'      prop\_values = '16.0  1.0 1.0  0.5      1.0    '    [../] |  |

We try to perform it in several sub-steps.

1. Remove “temp” from global parameters and leave it as variable only
2. We save PFDiffusionGrowth.c and PFDiffusionGrowth.h with different names PFDiffusionGrowthM1.\* and compile them without changes. The name in the script will change accordingly.
3. Next, we try to remove parameter for volumetric diffusion coefficient “Dvol” and add it as constant variable value

Table 11 Original C++-code for PFDiffusionGrowth.h and PFDiffusionGrowth.c

|  |  |
| --- | --- |
| #include "PFDiffusionGrowth.h"  registerMooseObject("CrowApp", PFDiffusionGrowth);  *template* <> InputParameters validParams<PFDiffusionGrowth>() {    InputParameters params = validParams<Material>();    params.addParam<Real>("Dvol", 0.01, "Volumetric diffusion ");    params.addParam<Real>("Dvap", 0.001, "Vapor Diffusion ");    params.addParam<Real>("Dsurf", 4, "surface diffusion");    params.addParam<Real>("Dgb", 0.4, "Grain Boundary diffusion");    params.addParam<Real>("kappa", 1.0,                          "The kappa multiplier for the interfacial energy");    params.addRequiredCoupledVar("rho", "phase field variable");    params.addRequiredCoupledVar("v", "array of order parameters");    return params;  }  PFDiffusionGrowth::PFDiffusionGrowth(const InputParameters &*parameters*)      : Material(parameters), \_Dvol(getParam<Real>("Dvol")),        \_Dvap(getParam<Real>("Dvap")), \_Dsurf(getParam<Real>("Dsurf")),        \_Dgb(getParam<Real>("Dgb")), \_kappa(getParam<Real>("kappa")),        \_rho(coupledValue("rho")), \_grad\_rho(coupledGradient("rho")),        \_v(coupledValue("v")),        \_D(declareProperty<Real>("D")),        // \_kappa\_c(declareProperty<Real>("kappa\_c")),        \_dDdc(declareProperty<Real>("dDdc")) {    // Array of coupled variables is created in the constructor    \_ncrys = coupledComponents(        "v"); // determine number of grains from the number of names passed in.    \_vals.resize(\_ncrys); // Size variable arrays    \_vals\_var.resize(\_ncrys);    // Loop through grains and load coupled variables into the arrays    for (*unsigned* *int* i = 0; i < \_ncrys; ++i) {      \_vals[i] = &coupledValue("v", i);      \_vals\_var[i] = coupled("v", i);    }  }  *void* PFDiffusionGrowth::computeQpProperties() {    Real SumEtaj = 0.0;    for (*unsigned* *int* i = 0; i < \_ncrys; ++i)      for (*unsigned* *int* j = 0; j < \_ncrys; ++j)        if (j != i)          SumEtaj += (\*\_vals[i])[\_qp] \*                     (\*\_vals[j])[\_qp]; // Sum all other order parameters    Real c = \_rho[\_qp];    c = c > 1.0 ? 1.0 : (c < 0.0 ? 0.0 : c);    Real phi = c \* c \* c \* (10 - 15 \* c + 6 \* c \* c);    phi = phi > 1.0 ? 1.0 : (phi < 0.0 ? 0.0 : phi);    \_D[\_qp] = \_Dvol \* phi + \_Dvap \* (1.0 - phi) + \_Dsurf \* c \* (1 - c) +              \_Dgb \* SumEtaj; // + \_Dvap\*(1 - phi) ;    Real dphidc = 30.0 \* c \* c \* (1 - 2 \* c + c \* c);    \_dDdc[\_qp] = \_Dvol \* dphidc - \_Dvap \* dphidc + \_Dsurf \* (1.0 - 2.0 \* c);    // \_kappa\_c[\_qp] = \_kappa;  } | #ifndef PFDIFFUSIONGROWTH\_H  #define PFDIFFUSIONGROWTH\_H  #include "Material.h"  //Forward Declarations  *class* PFDiffusionGrowth;  *template*<>  InputParameters validParams<PFDiffusionGrowth>();  *class* PFDiffusionGrowth : *public* Material  {  *public:*    PFDiffusionGrowth(const InputParameters & *parameters*);  *protected:*    virtual *void* computeQpProperties();  *private:*    std::vector<const VariableValue \*> \_vals;    std::vector<*unsigned* *int*> \_vals\_var;    Real \_Dvol;    Real \_Dvap;    Real \_Dsurf;    Real \_Dgb;    Real \_kappa;    const VariableValue & \_rho;    const VariableGradient & \_grad\_rho;    const VariableValue & \_v;    MaterialProperty<Real> & \_D;    // MaterialProperty<Real> & \_kappa\_c;    MaterialProperty<Real> & \_dDdc;  *unsigned* *int* \_ncrys;  };  #endif //PFDIFFUSIONGROWTH\_H |

## Intermediate step: checking MOOSE functions and Sudipta code.

### Test 1. Use output of one MaterailParsedFunction as an input for another function

We use code like the one shown in Table 12. Here function “chi” is defined in one DerivativeParsedMaterial block and used in another one.

Table (left) Example of using material properties in another material (right) Implementation of the DerivativeParsedMaterials in modified Sudipta code.

|  |  |
| --- | --- |
| [Materials]    [./Mobility]      type = *DerivativeParsedMaterial*      f\_name = Dchi      material\_property\_names = 'D chi'      function = 'D\*chi'      derivative\_order = 2    [../]    [./chi]      type = *DerivativeParsedMaterial*      f\_name = chi      material\_property\_names = 'Vm ha(etaa0,etab0,etad0) ka hb(etaa0,etab0,etad0) kb hd(etaa0,etab0,etad0) kd'      function = '(ha/ka + hb/kb + hd/kd) / Vm^2'      args = 'etaa0 etab0 etad0'      derivative\_order = 2    [../]  [] | [Materials]    [./sumofgr]      type = *DerivativeParsedMaterial*      f\_name = sumofgr      args = 'gr0 gr1'      function = (gr0^2+gr1^2)      derivative\_order = 2    [../]    [./chemical\_free\_energy]      type = *DerivativeParsedMaterial*      f\_name = Fc      args = 'c gr0 gr1'      constant\_names = 'A  B'      constant\_expressions = '16.0 1.0'      material\_property\_names = 'sumofgr'      function = A\*c^2\*(1-c)^2+B\*(c^2+6\*(1-c)\*sumofgr-4\*(2-c)\*(gr0^3+gr1^3)+3\*sumofgr^2)      derivative\_order = 2    [../]    [./phi]      type = *DerivativeParsedMaterial*      f\_name = phi      args = 'c'      function = 'c \* c \* c \* (10 - 15 \* c + 6 \* c \* c)'      derivative\_order = 1      outputs = exodus    [../]    [./mobility]      type = *DerivativeParsedMaterial*      f\_name = D      args = 'c gr0 gr1 temp'      constant\_names = 'Dvol Dvap Dsurf Dgb'      constant\_expressions = '0.01 0.001 4 0.4'      material\_property\_names = 'phi'      function = '(Dvol+0.0001\*temp)\*phi+Dvap\*(1.0-phi)+Dsurf\*c\*(1-c)+Dgb\*gr0\*gr1'      derivative\_order = 1      outputs = exodus    [../]  [] |

### Test 2. Using interpolation functions

Another important method if using realistic material properties is

We follow example shown in Table 13. As a first step we will introduce function “Dvol + 0.00001\*temp” as a DerivativeParsedMaterial. This is done in the code “sint1\_mater\_2gr\_el\_2d\_v25.i”. Next, we substitute this function with approximate piece-wise linear function using “type = *PiecewiseLinearInterpolationMaterial*” shown in Table 13.

Table Example of the code that uses linear interpolation as script function

|  |  |
| --- | --- |
| [Materials]    [./youngs\_modulus]      type = *PiecewiseLinearInterpolationMaterial*      x = '100 500'      y = '1e6 6e5'      property = youngs\_modulus      variable = temp    [../]    [./poissons\_ratio]      type = *PiecewiseLinearInterpolationMaterial*      x = '100 500'      y = '0   0.25'      property = poissons\_ratio      variable = temp    [../]    [./elasticity\_tensor]      type = *ComputeVariableIsotropicElasticityTensor*      args = temp      youngs\_modulus = youngs\_modulus      poissons\_ratio = poissons\_ratio    [../]  [] |  |

### Test 3. Read Material function in C-code

We use a fork of “sint1\_mater\_2gr\_el\_2d\_v24.i”. But now we try to access the value of Dv\_temp with c-code by using “getMaterialProperty”.

### Test 4. Reading IC from file.

We use code shown in Table 12.

Table Example of the code to read IC for phase-field from file.

|  |  |
| --- | --- |
| [Mesh]    type = *GeneratedMesh*    dim = 2    nx = 100    ny = 56    nz = 0    xmin = 0    xmax = 200    ymin = 0    ymax = 112    zmin = 0    zmax = 0  []  [GlobalParams]    op\_num = 6    var\_name\_base = gr  []  [Variables]    [./PolycrystalVariables]    [../]  []  [UserObjects]    [./circle\_IC]      type = *PolycrystalCircles*      file\_name = 'circles.txt'      read\_from\_file = true      execute\_on = 'initial'      threshold = 0.2      connecting\_threshold = 0.08      int\_width = 8    [../]  []  [ICs]    [./PolycrystalICs]      [./PolycrystalColoringIC]        polycrystal\_ic\_uo = circle\_IC      [../]    [../]  [] | [Kernels]    [./dt\_gr0]      type = *TimeDerivative*      variable = gr0    [../]    [./dt\_gr1]      type = *TimeDerivative*      variable = gr1    [../]    [./dt\_gr2]      type = *TimeDerivative*      variable = gr2    [../]    [./dt\_gr3]      type = *TimeDerivative*      variable = gr3    [../]    [./dt\_gr4]      type = *TimeDerivative*      variable = gr4    [../]    [./dt\_gr5]      type = *TimeDerivative*      variable = gr5    [../]  []  [Executioner]    type = *Transient*    scheme = bdf2    solve\_type = PJFNK    num\_steps = 0  []  [Outputs]    exodus = true    csv = false  [] |

Note. Reading grain parameters from file does not work properly. We do not see grain boundaries.

For example, we have the following working version

Table Working version of the SmoothCircleFromFileIC. Does not show bnds.

|  |  |  |
| --- | --- | --- |
| [Mesh]    type = *GeneratedMesh*    dim = 2    nx = 60    ny = 60    xmin = 4.0    xmax = 33.0    ymin = 8.0    ymax = 34.0    elem\_type = QUAD4  []  [GlobalParams]    displacements = 'disp\_x disp\_y'    var\_name\_base = gr    op\_num = 2.0    int\_width = 2.0  []  [Variables]    [./c]    [../]    [../]    [./w]    [../]    [./PolycrystalVariables]    [../]    [./disp\_x]      order = FIRST      family = LAGRANGE    [../]    [./disp\_y]      order = FIRST      family = LAGRANGE    [../]    [./temp]      initial\_condition = 400    [../]  []  [ICs]    [./IC\_c]      type = *SmoothCircleFromFileIC*      file\_name = 'TwoCircles.txt'      invalue = 0.95      outvalue = 0.05      variable = c      int\_width = 2      block = 0    [../]    # [./multip]    #   x\_positions = '12.6   24.7'    #   y\_positions = '22.4   24.1'    #   z\_positions = '0 0'    #   radii = '6.25 8.25'    #   # file\_name = 'TwoCircles.txt'    #   int\_width = 2.0    #   3D\_spheres = false    #   outvalue = 0.05    #   variable = c    #   invalue = 0.95    #   type = SpecifiedSmoothCircleIC    #   block = 0    # [../]  []  [AuxVariables]    [./bnds]    [../]    [./total\_en]      order = CONSTANT      family = MONOMIAL    [../]    [./unique\_grains]      order = CONSTANT      family = MONOMIAL    [../]    [./var\_indices]      order = CONSTANT      family = MONOMIAL    [../]    [./centroids]      order = CONSTANT      family = MONOMIAL    [../]      [./sigma11\_aux]      order = CONSTANT      family = MONOMIAL    [../]    [./sigma22\_aux]      order = CONSTANT      family = MONOMIAL    [../]  []  [Functions]    [./load]      type = *ConstantFunction*      value = 0.01    [../]  []  [Kernels]    [./dt\_gr0]      type = *TimeDerivative*      variable = gr0    [../]    [./dt\_gr1]      type = *TimeDerivative*      variable = gr1    [../]    [./heat]      type = *HeatConduction*      variable = temp    [../]    [./HeatSource]      type = *HeatSource*      function = '10\*exp(-((x-(31-2.7\*t))^2/2))\*exp(-abs(y-34)/1)'      variable = temp    [../]    [./TensorMechanics]      displacements = 'disp\_x disp\_y'    [../]    [./cres]      type = *SplitCHParsed*      variable = c      kappa\_name = kappa\_c      w = w      f\_name = F      args = 'gr0 gr1'    [../]    [./wres]      type = *SplitCHWRes*      variable = w      mob\_name = D    [../]    [./time]      type = *CoupledTimeDerivative*      variable = w      v = c    [../] | [./PolycrystalSinteringKernel]      c = c      consider\_rigidbodymotion = false      anisotropic = false      grain\_force = grain\_force      grain\_tracker\_object = grain\_center      grain\_volumes = grain\_volumes      translation\_constant = 10.0      rotation\_constant = 1.0    [../]  []  [BCs]  # Boundary Condition block    [./Periodic]      [./left\_right]        auto\_direction = 'x      [../]    [../]    [./bottom]      type = *DirichletBC*      variable = temp      boundary = bottom      value = 700    [../]  []  [Materials]    [./sumofgr]      type = *DerivativeParsedMaterial*      f\_name = sumofgr      args = 'gr0 gr1'      function = (gr0^2+gr1^2)      derivative\_order = 2    [../]    [./chemical\_free\_energy]      type = *DerivativeParsedMaterial*      f\_name = Fc      args = 'c gr0 gr1'      constant\_names = 'A  B'      constant\_expressions = '16.0 1.0'      material\_property\_names = 'sumofgr'      function = A\*c^2\*(1-c)^2+B\*(c^2+6\*(1-c)\*sumofgr-4\*(2-c)\*(gr0^3+gr1^3)+3\*sumofgr^2)      derivative\_order = 2    [../]    # [./Dv]    [./CH\_mat]      type = *PFDiffusionGrowthM3*      block = 0      Dvol = 0.01      rho = c      T = temp      v = 'gr0 gr1'      outputs = console    [../]    [./constant\_mat]      type = *GenericConstantMaterial*      block = 0      prop\_names = '  A    B   L    kappa\_op kappa\_c'      prop\_values = ' 16.0 1.0 1.0  0.5      1.0    '    [../]    [./elasticity\_tensor]      type = *ComputeElasticityTensor*      block = 0      # lambda, mu values      C\_ijkl = '7 7'        fill\_method = symmetric\_isotropic    [../]    [./stress]      type = *ComputeLinearElasticStress*      block = 0    [../]    [./strain]      type = *ComputeSmallStrain*      # block = 0      displacements = 'disp\_x disp\_y'      eigenstrain\_names = eigenstrain    [../]    [./elastic\_free\_energy]      type = *ElasticEnergyMaterial*      f\_name = Fe      block = 0      args = 'c'      derivative\_order = 2    [../]    [./free\_energy]      type = *DerivativeSumMaterial*      block = 0      f\_name = F      sum\_materials = 'Fc Fe'      args = 'c'      derivative\_order = 2    [../]    [./thermal\_strain]      type = *ComputeThermalExpansionEigenstrain*      block = 0      temperature = temp      stress\_free\_temperature = 400      thermal\_expansion\_coeff = 1e-8      eigenstrain\_name = eigenstrain    [../]    [./heat]      type = *HeatConductionMaterial*      block = 0      specific\_heat = 1.0      thermal\_conductivity = 1.0    [../]    [./poissons\_ratio]      type = *PiecewiseLinearInterpolationMaterial*      x = '100 500'      y = '0   0.25'      property = poissons\_ratio      variable = temp    [../]  []  [VectorPostprocessors]    [./grain\_volumes]      type = *FeatureVolumeVectorPostprocessor*      flood\_counter = grain\_center      execute\_on = 'initial timestep\_begin'    [../]  []  [UserObjects]    [./grain\_center]      type = *GrainTracker*      outputs = none      compute\_var\_to\_feature\_map = true      execute\_on = 'initial timestep\_begin'    [../]  []  [AuxKernels]    [./bnds]      type = *BndsCalcAux*      variable = bnds      v = 'gr0 gr1 '    [../]    [./Total\_en]      type = *TotalFreeEnergy*      variable = total\_en      kappa\_names = 'kappa\_c kappa\_op kappa\_op '      interfacial\_vars = 'c  gr0 gr1 '    [../] | [./unique\_grains]      type = *FeatureFloodCountAux*      variable = unique\_grains      flood\_counter = grain\_center      field\_display = UNIQUE\_REGION      execute\_on = timestep\_begin    [../]    [./var\_indices]      type = *FeatureFloodCountAux*      variable = var\_indices      flood\_counter = grain\_center      field\_display = VARIABLE\_COLORING      execute\_on = timestep\_begin    [../]    [./centroids]      type = *FeatureFloodCountAux*      variable = centroids      execute\_on = timestep\_begin      field\_display = CENTROID      flood\_counter = grain\_center    [../]  [./matl\_sigma11]      type = *RankTwoAux*      rank\_two\_tensor = stress      index\_i = 0      index\_j = 0      variable = sigma11\_aux    [../]    [./matl\_sigma22]      type = *RankTwoAux*      rank\_two\_tensor = stress      index\_i = 1      index\_j = 1      variable = sigma22\_aux    [../]  [Postprocessors]    [./elem\_c]      type = *ElementIntegralVariablePostprocessor*      variable = c    [../]    [./elem\_bnds]      type = *ElementIntegralVariablePostprocessor*      variable = bnds    [../]    [./total\_energy]      type = *ElementIntegralVariablePostprocessor*      variable = total\_en    [../]    [./free\_en]      type = *ElementIntegralMaterialProperty*      mat\_prop = F    [../]    [./el\_free\_en]      type = *ElementIntegralMaterialProperty*      mat\_prop = Fe    [../]    [./ch\_free\_en]      type = *ElementIntegralMaterialProperty*      mat\_prop = Fc    [../]    [./dofs]      type = *NumDOFs*      system = 'NL'    [../]    [./tstep]      type = *TimestepSize*    [../]    [./int\_area]      type = *InterfaceAreaPostprocessor*      variable = c    [../]    [./grain\_size\_gr0]      type = *ElementIntegralVariablePostprocessor*      variable = gr0    [../]    [./grain\_size\_gr1]      type = *ElementIntegralVariablePostprocessor*      variable = gr1    [../]    [./gb\_area]      type = *GrainBoundaryArea*    [../]  []  [Preconditioning]    [./SMP]      type = *SMP*      coupled\_groups = 'c,w c,gr0,gr1 '    [../]  []  [Executioner]    type = *Transient*    scheme = BDF2    solve\_type = 'PJFNK'    petsc\_options\_iname = '-pc\_type -ksp\_grmres\_restart -sub\_ksp\_type -sub\_pc\_type -pc\_asm\_overlap'    petsc\_options\_value = 'asm         31   preonly   lu      1'    l\_max\_its = 15    nl\_max\_its = 15    nl\_abs\_tol = 1e-04    nl\_rel\_tol = 1e-04    l\_tol = 1e-04    end\_time = 10    #dt = 0.01    [./TimeStepper]      type = *IterationAdaptiveDT*      dt = 0.05      growth\_factor = 1.15    [../]  []  [Adaptivity]    marker = bound\_adapt    max\_h\_level = 2    [./Indicators]      [./error]        type = *GradientJumpIndicator*        variable = bnds      [../]    [../]    [./Markers]      [./bound\_adapt]        type = *ValueRangeMarker*        lower\_bound = 0.01        upper\_bound = 0.99        variable = bnds      [../]    [../]  []  [Outputs]    print\_linear\_residuals = true    csv = true    # exodus = true    gnuplot = true    print\_perf\_log = true    [./console]      type = *Console*      perf\_log = true    [../]    [./exodus]      type = *Exodus*      elemental\_as\_nodal = true    [../]  [] |

## Adding realistic parameters for the diffusion and energy functions

We put on hold reading grain parameters from the file and proceed with

# References

1. Biswas, S., Schwen, D., Singh, J., & Tomar, V. (2016). A study of the evolution of microstructure and consolidation kinetics during sintering using a phase field modeling based approach. *Extreme Mechanics Letters*, *7*, 78–89. https://doi.org/10.1016/j.eml.2016.02.017
2. Biswas, S., Schwen, D., & Tomar, V. (2018). Implementation of a phase field model for simulating evolution of two powder particles representing microstructural changes during sintering. *Journal of Materials Science*, *53*(8), 5799–5825. https://doi.org/10.1007/s10853-017-1846-3
3. C. Minkwitz, and C. H. J. S. L. (1997). Grain Boundary Self-Diffusion in Polycrystalline Tungsten at Low Temperatures. *Physica Status Solidi (B)*, *202*(2), 931–940. https://doi.org/10.1002/1521-3951(199708)202:2<931::AID-PSSB931>3.0.CO;2-O
4. Chockalingam, K., Kouznetsova, V. G., van der Sluis, O., & Geers, M. G. D. (2016). 2D Phase field modeling of sintering of silver nanoparticles. *Computer Methods in Applied Mechanics and Engineering*, *312*(2016), 492–508. https://doi.org/10.1016/j.cma.2016.07.002
5. Dzepina, B., Balint, D., & Dini, D. (2019). Journal of the European Ceramic Society A phase fi eld model of pressure-assisted sintering. *Journal of the European Ceramic Society*, *39*(2–3), 173–182. https://doi.org/10.1016/j.jeurceramsoc.2018.09.014
6. Grujicic, M., Zhao, H., & Krasko, G. L. (1999). *Atomistic Simulation of Sigma 3 (111) Grain Boundary Fracture in Tungsten Containing Various Impurities*.
7. Yang, Q., Kirshtein, A., Ji, Y., Liu, C., Shen, J., & Chen, L. Q. (2018). A thermodynamically consistent phase‐field model for viscous sintering. *Journal of the American Ceramic Society*, *102*(2), 674–685. https://doi.org/10.1111/jace.16021
8. Yang, Y., Kühn, P., Yi, M., Egger, H., & Xu, B. X. (2020). Non-isothermal Phase-Field Modeling of Heat–Melt–Microstructure-Coupled Processes During Powder Bed Fusion. *Jom*. https://doi.org/10.1007/s11837-019-03982-y
9. Yang, Y., Ragnvaldsen, O., Bai, Y., Yi, M., & Xu, B. X. (2019). 3D non-isothermal phase-field simulation of microstructure evolution during selective laser sintering. *Npj Computational Materials*, *5*(1). https://doi.org/10.1038/s41524-019-0219-7
10. Zhang, X., & Liao, Y. (2018). A phase-field model for solid-state selective laser sintering of metallic materials. *Powder Technology*, *339*, 677–685. https://doi.org/10.1016/j.powtec.2018.08.025

1. you may have very poor scaling between variables in a multi-physics simulation. You may even have run into issues if you have nodal boundary conditions (which introduce values of unity on the diagonals) and the Jacobian entries from your physics (kernels) are very large. You want your condition number to be as close to unity as possible. To address the latter problem or poor relative scaling between variables, you can use MOOSE's automatic scaling feature which will bring different physics Jacobians as close to unity as possible. To turn on this feature, set the automatic\_scaling parameter in the Executioner block to true. Additionally, if you want to update scaling factors at every time step then set Executioner/compute\_scaling\_once=false. By default this latter parameter is set to true in order to save computational expense. [↑](#footnote-ref-1)
2. Heat source term [↑](#footnote-ref-2)
3. **NEWTON** - Direct solution of the system of equations using Newton's method. The full and accurate Jacobian is required. Thus, the [Preconditioner](http://mooseframework.org/wiki/MooseSystems/Preconditioners/) block must be employed for systems with multiple nonlinear variables.

   **JFNK** - The system is solved using Jacobian Free Newton Krylov (JFNK), so no Jacobian terms are needed. However, JFNK often does not perform well without preconditioning.

   **PJFNK** - The system is solved using preconditioned JFNK. The Jacobian is used to precondition the matrix, but it does not have to be fully correct and can neglect off-diagonal terms. [↑](#footnote-ref-3)