Introduction to the phase field method and the MOOSE phase field module

Larry Aagesen[a], Andrea Jokisaari[a], Daniel Schwen[a], Michael Tonks[b]

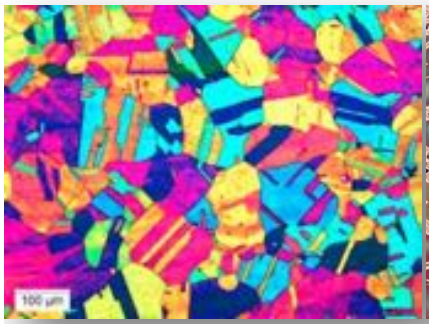
[a]Computational Mechanics and Materials Department
Idaho National Laboratory
andrea.jokisaari@inl.gov, larry.aagesen@inl.gov, daniel.schwen@inl.gov

[b]Department of Materials Science and Engineering
University of Florida
michael.tonks@ufl.edu

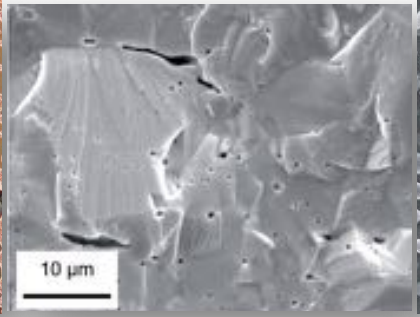
MOOSE mooseframework.org

The purpose of the phase field method is to model microstructure evolution

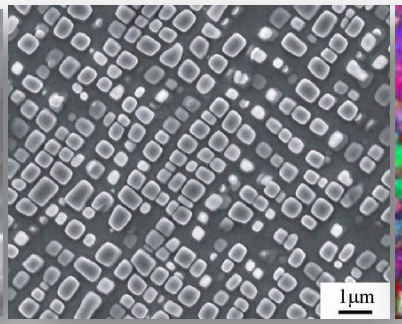
- The microstructure is established during fabrication


Annealed Stainless Steel

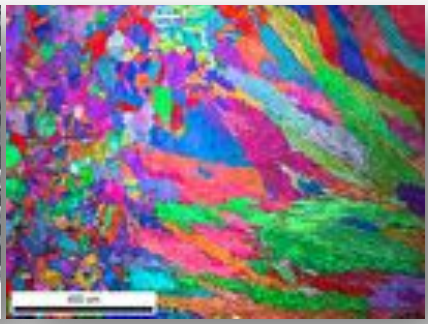

Cast Bronze



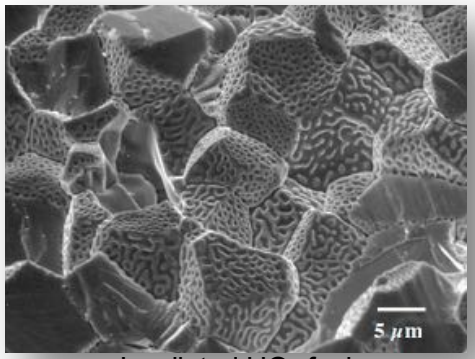
Sintered UO2



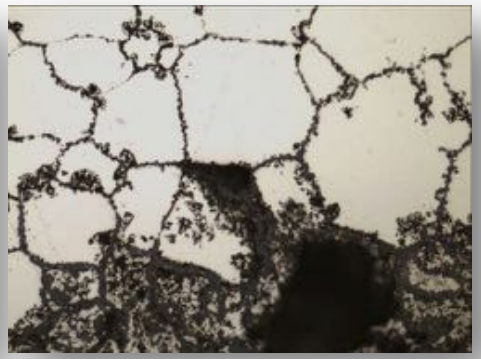
Co-Al-W Superalloy

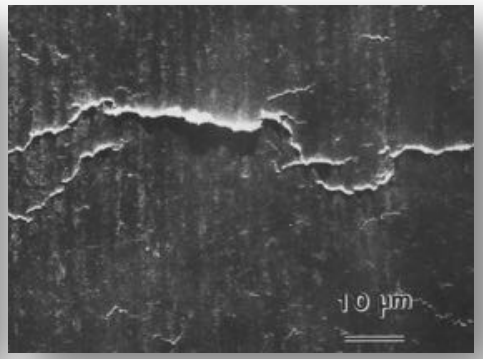

Friction stir welded steel

- It can continue to evolve during operation

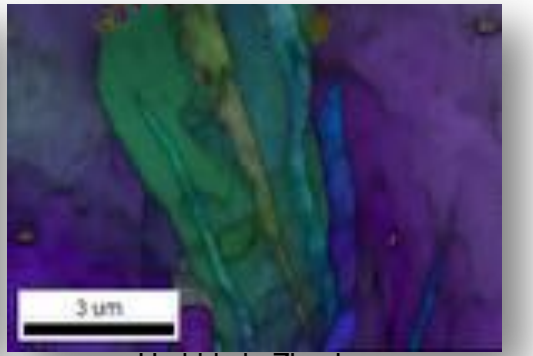


Irradiated UO_2 fuel


Corrosion in stainless steel



Micro-cracking in steel

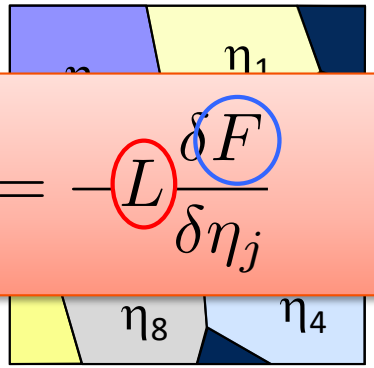


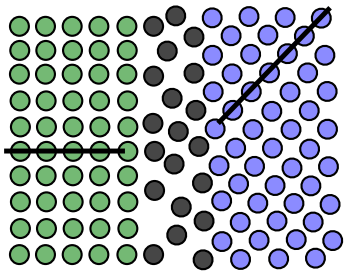
Hydride in Zircaloy

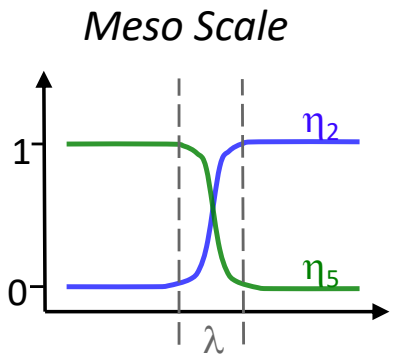
In the phase field method, the microstructure is described by continuous variables

- Non-conserved order parameters describe the atomic order

$$\frac{\partial \eta_j}{\partial t} = -L\frac{\delta F}{\delta \eta_j}$$

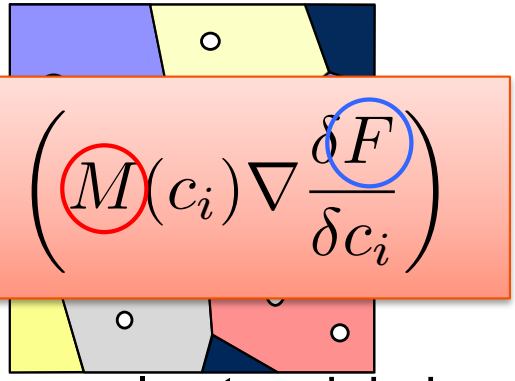


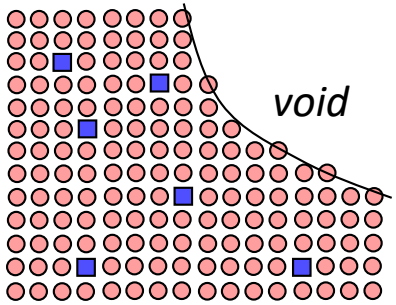


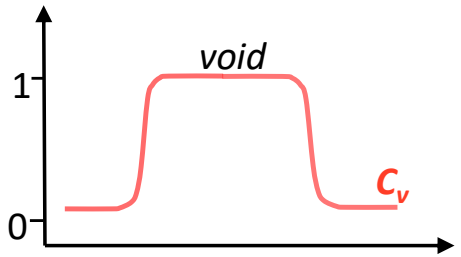


- Conserved concentrations describe atom concentrations

$$\frac{\partial c_i}{\partial t} = \nabla \cdot \left(M(c_i) \nabla \frac{\delta F}{\delta c_i} \right)$$

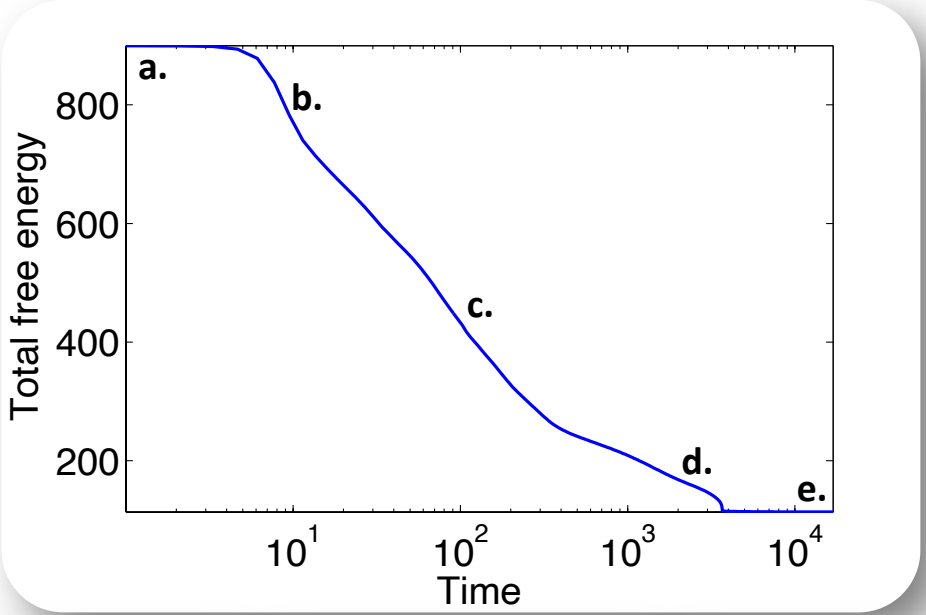




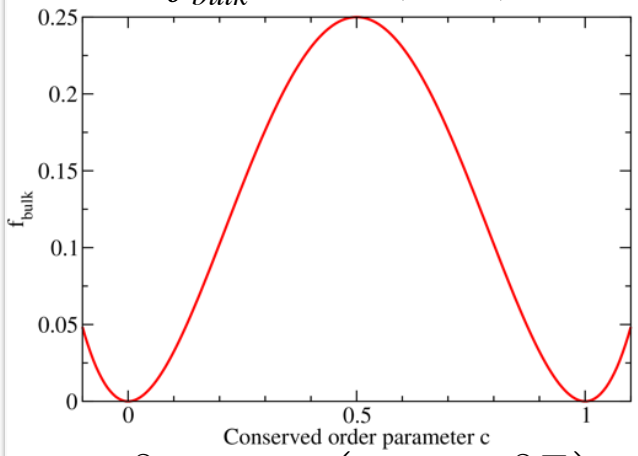


- The variables evolve to minimize a functional defining the free energy

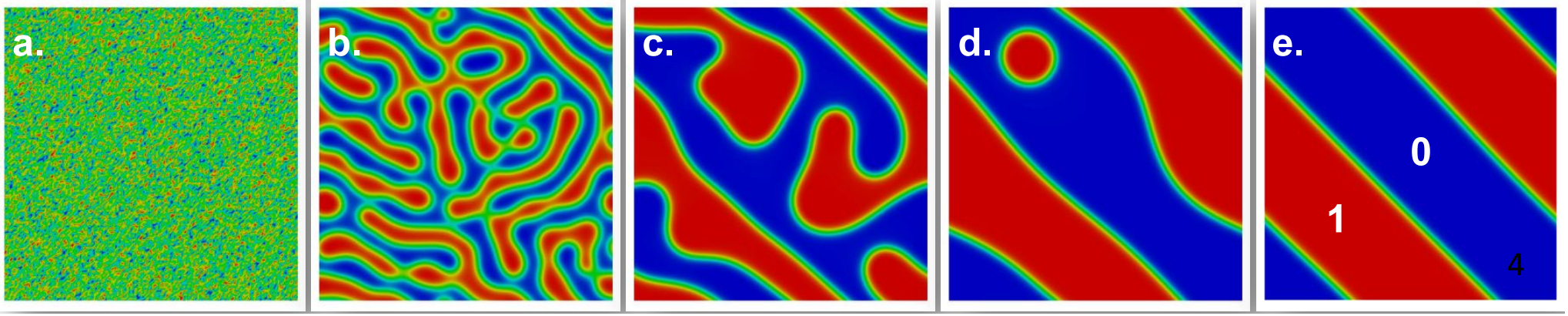
Spinodal Decomposition Example: the microstructure evolves from its initial state to lower the free energy



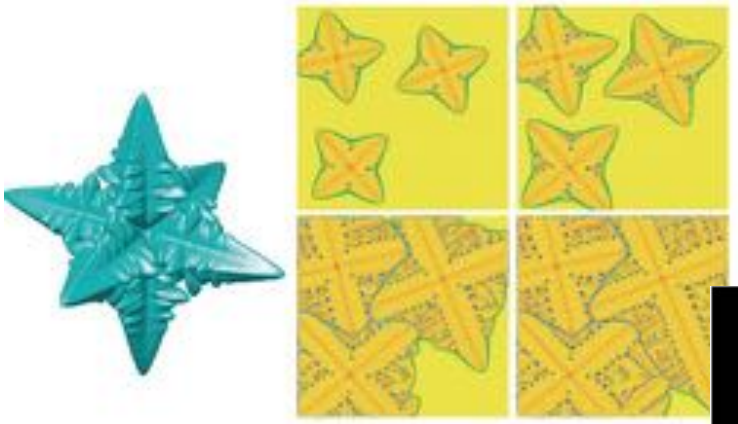
$$f_{bulk} = 4c^2(1-c)^2$$



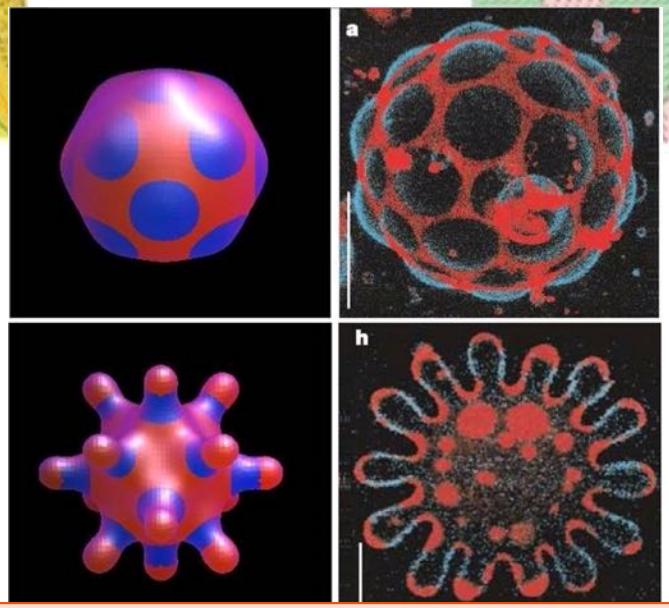
$$\frac{\partial c_i}{\partial t} = \nabla \cdot \left(M(c_i) \nabla \frac{\partial F}{\partial c_i} \right)$$

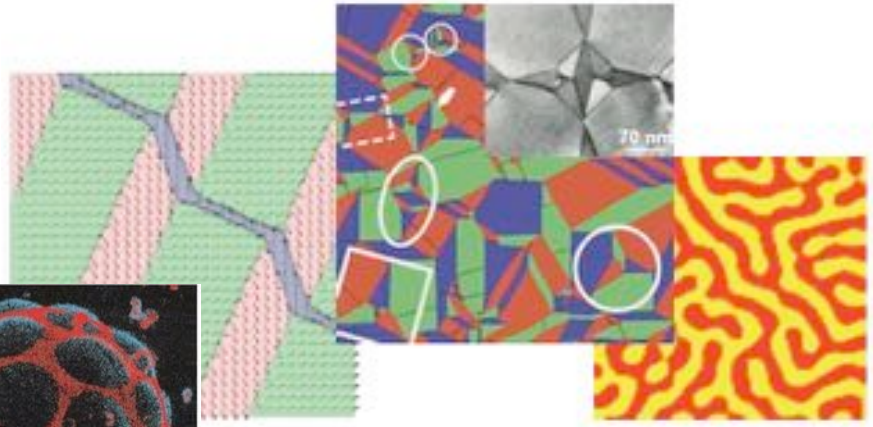


Phase Field Has Been Used in Many Areas

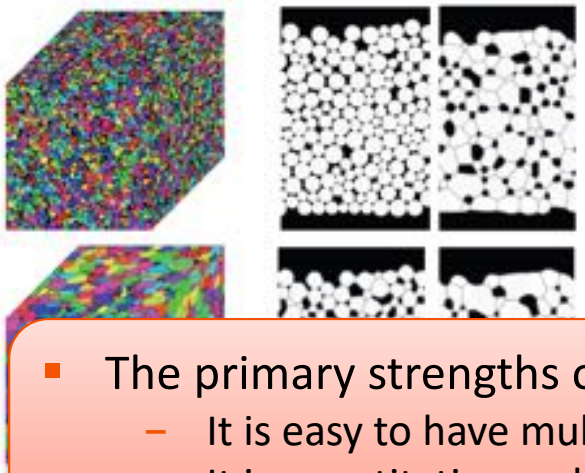


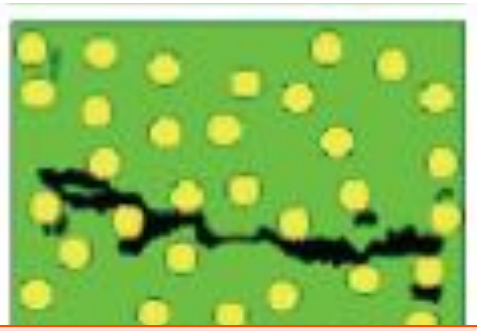
solidification (dendrite growth)





phase transformations

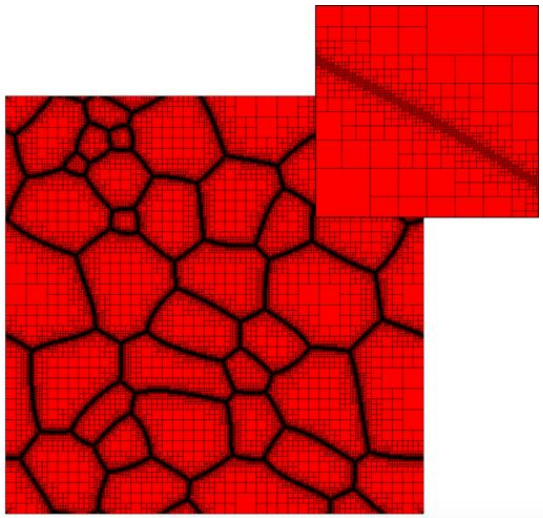


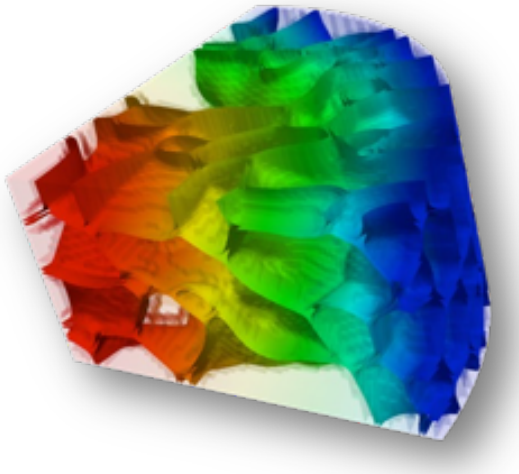


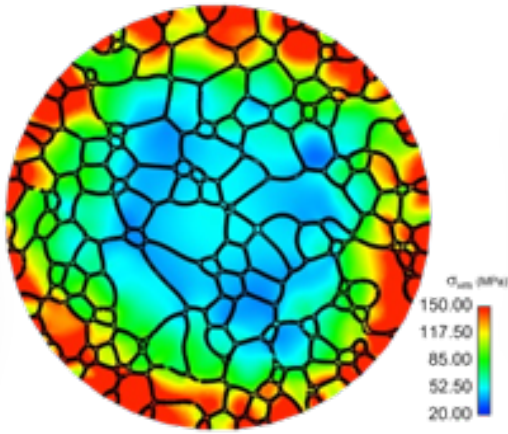
- The primary strengths of the phase field method are:
  - It is easy to have multiple physical phenomenon occurring at once, including mechanics
  - It is quantitative and can represent real materials

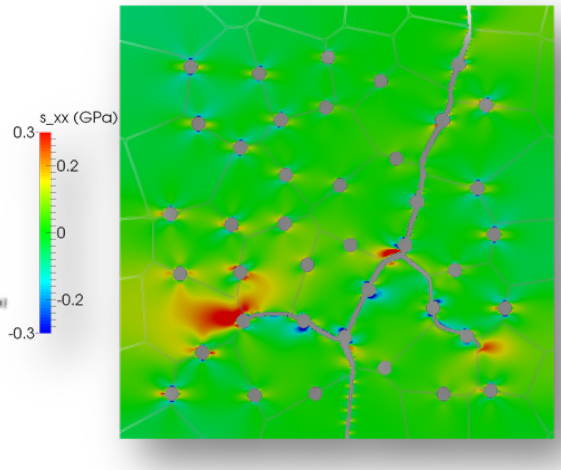
The phase field PDEs can be solved in various ways, but we use the finite element method

- Methods include finite difference, finite volume, FFT, and FEM.
- Benefits of FEM (MOOSE)
  - Can model any domain shape
  - Can employ a large range of different boundary conditions
  - Scales well up to large numbers of processors
  - Can easily couple to multiple physics





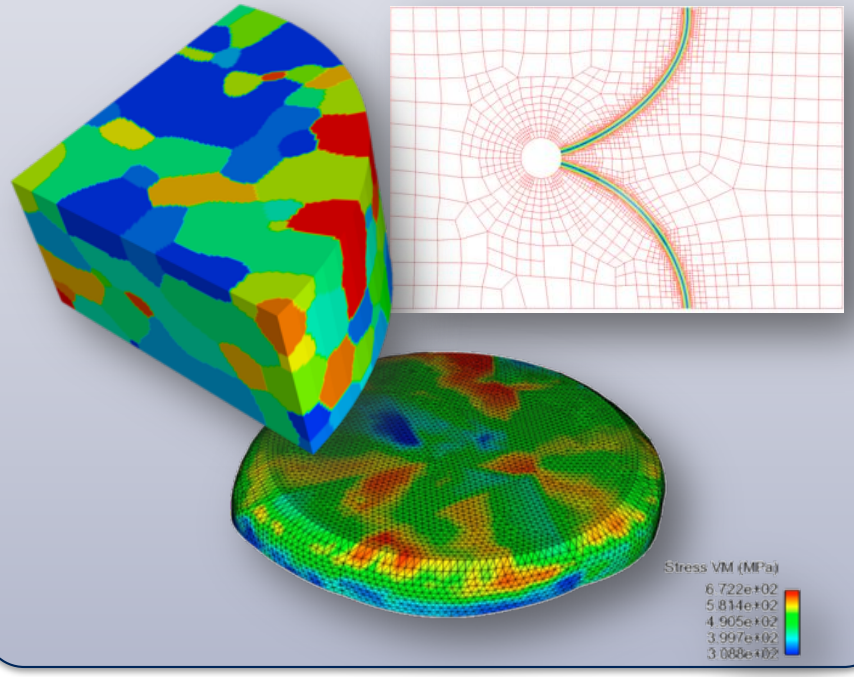




FEM with MOOSE has great flexibility in domain shape and boundary conditions

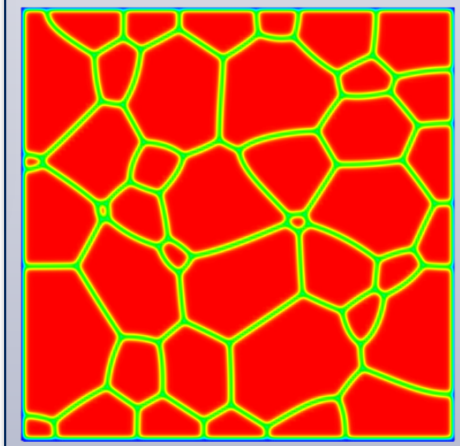
Flexible Domain Shapes

- With FEM, you can model any domain that you can mesh. Therefore, nearly any domain shape can be modeled.
- You can also mesh microstructural features, such as grains or pores

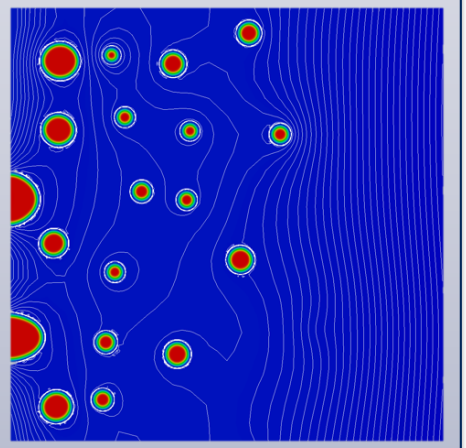


Flexible Boundary Conditions

- A large range of different boundary conditions are available in MOOSE
  - Periodic BCs
  - Dirichlet BCs (specified values on boundary or defined by function)
  - Neumann BCs (integrated BC in residual, often used to define flux)
- Users can add new boundary conditions



Φ_i = 0 on outer boundary

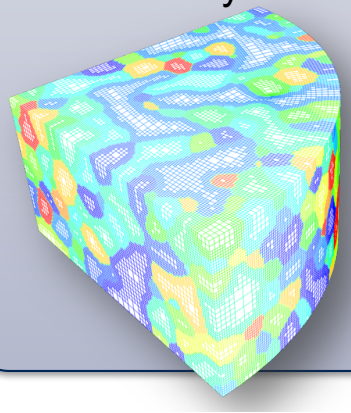


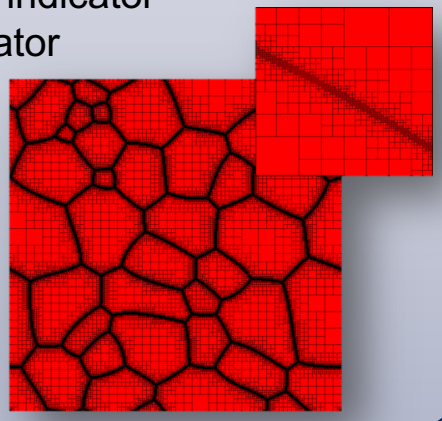
Positive flux on left, negative on right

Any model implemented with MOOSE has access to mesh and time step adaptivity

Mesh Adaptivity

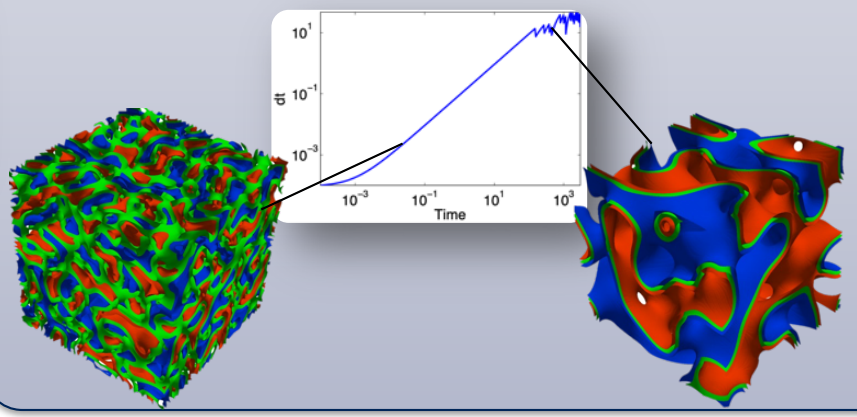
- Requires no code development
- Refinement or coarsening is defined by a marker that be related to
  - An error estimator
  - Variable values
  - Stipulated by some other model
- Error indicators include the
  - Gradient jump indicator
  - Flux jump indicator
  - Laplacian jump indicator
  - Analytical indicator





Transient Time Step Adaptivity

- The time step in transient simulations can change with time
- Various time steppers exist to define dt:
  - Defined by a function
  - Adapts to maintain consistent solution behavior
  - Adapts to maintain consistent solution time
- Users can write new time steppers



Solving the phase field equations is a boundary value problem with PDEs

- The equations are:
  - Allen-Cahn $\frac{\partial \eta_j}{\partial t} = -L_j \frac{\delta F}{\eta_j}$
  - Cahn-Hilliard $\frac{\partial c_i}{\partial t} = -\nabla \cdot M_i \nabla \frac{\delta F}{c_i}$
  - The initial conditions define the initial state of the microstructure
  - The boundary conditions depend on the problem being solved
- To solve the PDE in time, we use implicit time integration
  - $c_i(t + dt) = c_i(t) + f(c_i(t + dt))$
  - We solve the nonlinear equation using Newton's method or similar methods

The phase field variables evolve as dictated by functional derivatives of a free energy function

- The symbol δ indicates the functional derivative

$$\frac{\partial \eta_j}{\partial t} = -L_j \frac{\delta F}{\eta_j} \qquad \frac{\partial c_i}{\partial t} = -\nabla \cdot M_i \nabla \frac{\delta F}{c_i}$$

- The free energy functional includes the local energy (thermodynamic), deformation energy and gradient energy terms

$$F = \int_V \left(f_{loc}(c_i, \eta_j, \ldots, T) + E_d(c_i, \eta_j) + \sum_i \frac{\kappa_i}{2} (\nabla c_i)^2 + \sum_j \frac{\kappa_j}{2} (\nabla \eta_j)^2 \right) dV$$

Here is more information regarding the functional derivative

- From Wikipedia, "functional derivative":
  The functional (or variational) derivative relates a change in a functional to a change in a function that the functional depends on.
  - Let $F = \int f(r, c, \nabla c) dV$, where F is the total free energy and f is the free energy density
  - $\frac{\delta F}{\delta c} = \frac{\partial f}{\partial c} - \nabla \cdot \frac{\partial f}{\partial \nabla c}$
- So, the derivatives of our free energy functional are

$$F = \int_V \left(f_{loc}(c_i, \eta_j, \ldots, T) + E_d(c_i, \eta_j) + \sum_i \frac{\kappa_i}{2} (\nabla c_i)^2 + \sum_j \frac{\kappa_j}{2} (\nabla \eta_j)^2 \right) dV$$

$$\frac{\delta F}{\delta \eta_j} = \left(\frac{\partial f_{loc}}{\partial \eta_j} + \frac{\partial E_d}{\partial \eta_j} - \nabla \cdot \kappa_j \nabla \eta_j \right)$$

$$\frac{\delta F}{\delta c_i} = \left(\frac{\partial f_{loc}}{\partial c_i} + \frac{\partial E_d}{\partial c_i} - \nabla \cdot \kappa_i \nabla c_i \right)$$

The basic phase field equations have local, deformation, and gradient contributions

- Allen-Cahn: $\frac{\partial \eta_j}{\partial t} = -L_j \left(\frac{\partial f_{loc}}{\partial \eta_j} + \frac{\partial E_d}{\partial \eta_j} - \nabla \cdot \kappa_j \nabla \eta_j \right)$
- Cahn-Hilliard: $\frac{\partial c_i}{\partial t} = \nabla \cdot M_i \nabla \left(\frac{\partial f_{loc}}{\partial c_i} + \frac{\partial E_d}{\partial c_i} - \nabla \cdot \kappa_i \nabla c_i \right)$
- The form of the equations are fairly standard in most phase field models
- Phase field models differ due to their
  - Energy terms: f_{loc}, E_d,
  - Interfacial parameters: κ_j, κ_i
  - Kinetic terms: L_j, M_i

The phase field module simplifies the development of new phase field models

- Kernels that implement the weak form of the phase field equations have been created in the phase field module: https://mooseframework.org/modules/phase_field/index.html
- The correct kernels are automatically created when non-conserved and conserved variables are defined
- The required free energy functions and interfacial and kinetic parameters are defined by the user as materials
  - Energy terms: f_{loc}, E_d,
  - Interfacial parameters: κ_j, κ_i
  - Kinetic terms: L_j, M_i

All MOOSE phase field simulations need four types of objects

Variables

c_i, η_j — *Define non-conserved and conserved quantities to represent the microstructure*

```
[Modules]
  [./PhaseField]
    [./Nonconserved]
      ...
    [../]
  [../]
[]
```

```
[Modules]
  [./PhaseField]
    [./Conserved]
      ...
    [../]
  [../]
[]
```

ICs (Initial Conditions)

Defines the initial microstructure, as defines by the initial variable values

```
[ICs]
  ...
[]
```

Free energy, material properties

L, M, κ, f_{loc} — *Are used in the kernels, but are material specific. Can be functions of other variables.*

```
[Materials]
  ...
[]
```

Boundary Conditions

L, M, κ, f_{loc} — *Define the state of the material at the domain boundaries, e.g. $c_i = 0$ at $x = 0$. Could be periodic or involve gradients*

```
[BCs]
  ...
[]
```

Non-conserved variables are created in the phase field module block

- $\frac{\partial \eta_j}{\partial t} = -L_j \left(\frac{\partial f_{loc}}{\partial \eta_j} + \frac{\partial E_d}{\partial \eta_j} - \nabla \cdot \kappa_j \nabla \eta_j \right)$
- This block creates a nonconserved variable called eta and all the corresponding kernels
  - $f_{loc} + E_d$ is stored in a material property called F
  - κ_j is stored in a property called kappa_eta
  - L_j is stored in a property called L

```
[Modules]
  [./PhaseField]
    [./Nonconserved]
      [./eta]
        free_energy = F
        kappa = kappa_eta
        mobility = L
      [../]
    [../]
  [../]
[]
```

Conserved variables are also created in the phase field module block

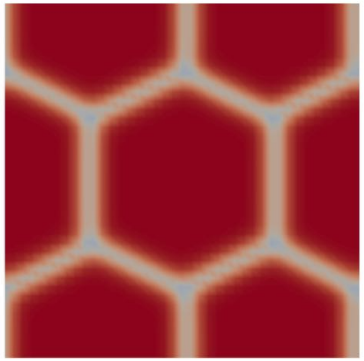
- $\frac{\partial c_i}{\partial t} = \nabla \cdot M_i \nabla \left(\frac{\partial f_{loc}}{\partial c_i} + \frac{\partial E_d}{\partial c_i} - \nabla \cdot \kappa_i \nabla c_i \right)$, however we often split it into two
- $\frac{\partial c_i}{\partial t} = \nabla \cdot M_i \nabla (\mu_i)$ and $\mu_i = \frac{\partial f_{loc}}{\partial c_i} + \frac{\partial E_d}{\partial c_i} - \nabla \cdot \kappa_i \nabla c_i$
- This block creates conserved variable c and the corresponding kernels, and it splits the equations, if desired
  - $f_{loc} + E_d$ is stored in a material property called F
  - κ_i is stored in a property called kappa_c
  - M_i is stored in a property called M
  - Solve_type determines if the equation is split
    - direct: No split (3rd order elements)
    - forward_split: New variable chem_pot_c is created
    - backward_split: Same, but residuals reversed

```
[Modules]
  [./PhaseField]
    [./Conserved]
      [./c]
        free_energy = F
        kappa = kappa_c
        mobility = M
        solve_type = forward_split
      [../]
    [../]
  [../]
[]
```

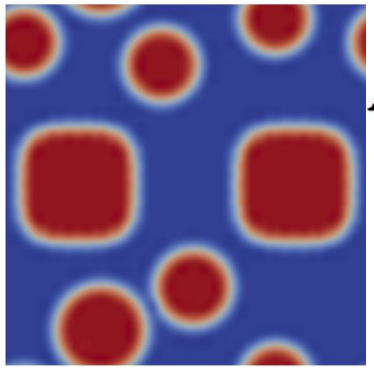
The initial stage of the microstructure is defined by the variable ICs

- There are many different ICs available in MOOSE to define your initial microstructure

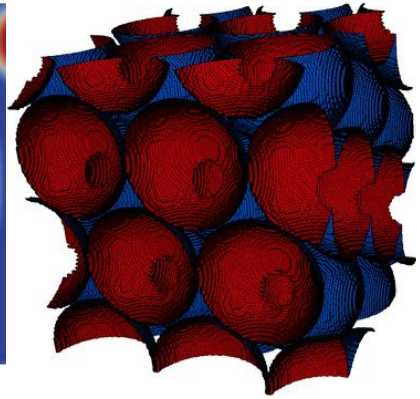
PolycrystalHexGrainIC



BimodalSuperellipsoidsIC

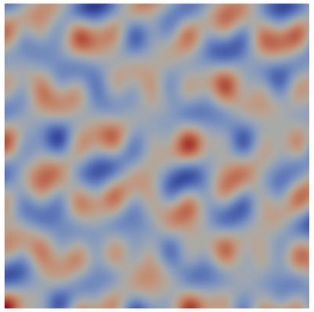


ClosePackIC

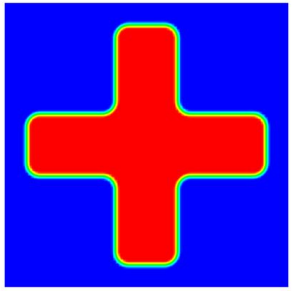


RndBoundingBoxIC

FunctionIC



CrossIC



```
[ICs]
  [./eta_IC]
    type = SmoothCircleIC
    variable = eta
    x1 = 200
    y1 = 200
    radius = 20
    int_width = 7
    invalue = 1
    outvalue = 0.0065
  [../]
[]
```

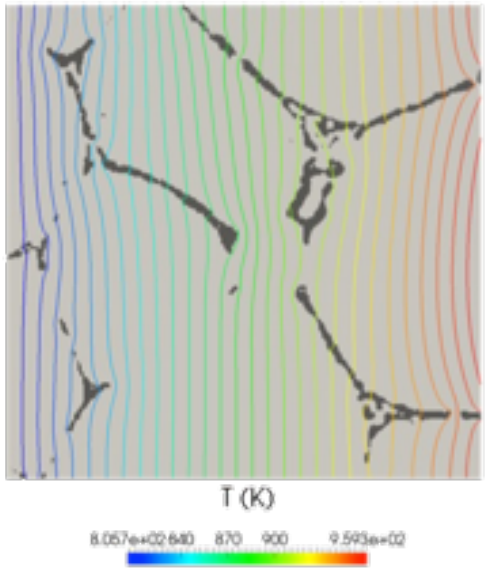
The ImageFunction can be used to create any IC you like from an image

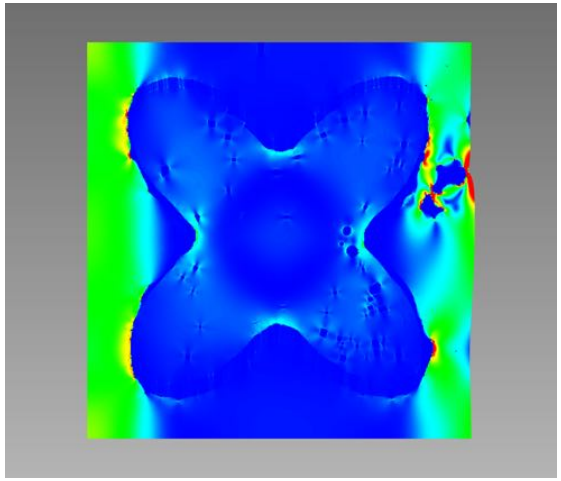
```
[Functions]
  [./image_func]
    type = ImageFunction
    file = Example.png
  [../]
[]
```

```
[Mesh]
  type = ImageFunction
  file = Example.png
  dim = 2
[]
```

```
[ICs]
  [./var_ic]
    type = FunctionIC
    function = image_func
    variable = var
  [../]
[]
```

- Values can be read in directly from the color intensity or you can use a cut-off threshold





We need many derivatives of material properties for residuals and Jacobians

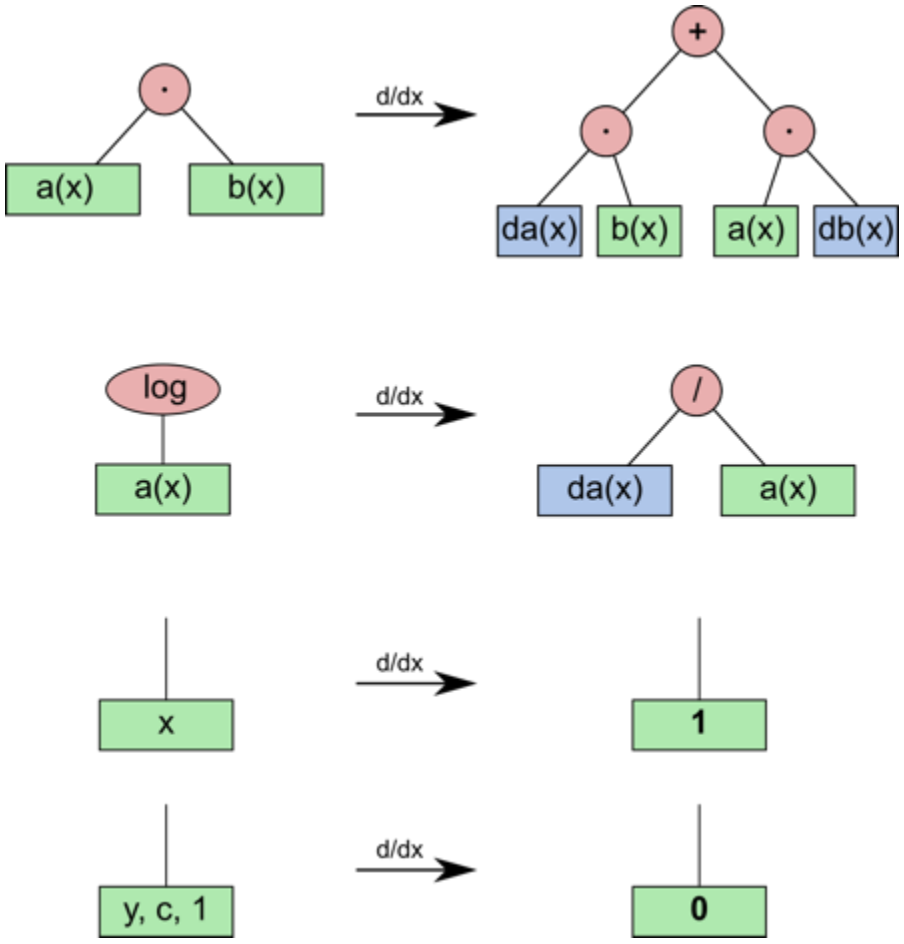
- Allen-Cahn: $\frac{\partial \eta_j}{\partial t} = -L_j \left(\frac{\partial f_{loc}}{\partial \eta_j} + \frac{\partial E_d}{\partial \eta_j} - \nabla \cdot \kappa_j \nabla \eta_j \right)$
  - $\frac{\partial f_{loc}}{\partial \eta_j}, \frac{\partial^2 f_{loc}}{\partial \eta_j^2}, \frac{\partial E_d}{\partial \eta_j}, \frac{\partial^2 E_d}{\partial \eta_j^2}, \frac{\partial L_j}{\partial \eta_j}, \ldots$
- Cahn-Hilliard: $\frac{\partial c_i}{\partial t} = \nabla \cdot M_i \nabla \left(\frac{\partial f_{loc}}{\partial c_i} + \frac{\partial E_d}{\partial c_i} - \nabla \cdot \kappa_i \nabla c_i \right)$
  - $\nabla \frac{\partial f_{loc}}{\partial c_i} = \frac{\partial^2 f_{loc}}{\partial c_i^2} \nabla c_i, \frac{\partial^3 f_{loc}}{\partial c_i^3}, \nabla \frac{\partial E_d}{\partial c_i} = \frac{\partial^2 E_d}{\partial c_i^2} \nabla c_i, \frac{\partial^3 E_d}{\partial c_i^3}, \frac{\partial M_i}{\partial c_i}, \ldots$
- Split Cahn-Hilliard: $\frac{\partial c_i}{\partial t} = \nabla \cdot M_i \nabla (\mu_i)$ and $\mu_i = \frac{\partial f_{loc}}{\partial c_i} + \frac{\partial E_d}{\partial c_i} - \nabla \cdot \kappa_i \nabla c_i$
  - $\frac{\partial f_{loc}}{\partial c_i}, \frac{\partial^2 f_{loc}}{\partial c_i^2}, \frac{\partial E_d}{\partial c_i}, \frac{\partial^2 E_d}{\partial c_i^2}, \frac{\partial M_i}{\partial c_i}, \ldots$

DerivativeParsedMaterial *in MOOSE defines a material property and its derivatives*

- Each MOOSE Material class can provide **multiple Material Properties**
- DerivativeParsedMaterial provides a well defined set of Material Properties
  - A function value, stored in the material property f_name
  - All necessary derivatives with respect to the variables f_name depends on
- The derivatives are regular Material Properties with an enforced naming convention
  - Example F, dF/dc, d^2F/dc^2, dF/deta, d^2F/dcdeta …
  - You don't need to know the property names besides f_name, unless you want to look at them in the output!
- The material property f_name could be used as a free energy, mobility, or whatever else you need

You define your functions directly in the input file and the derivatives are taken automatically

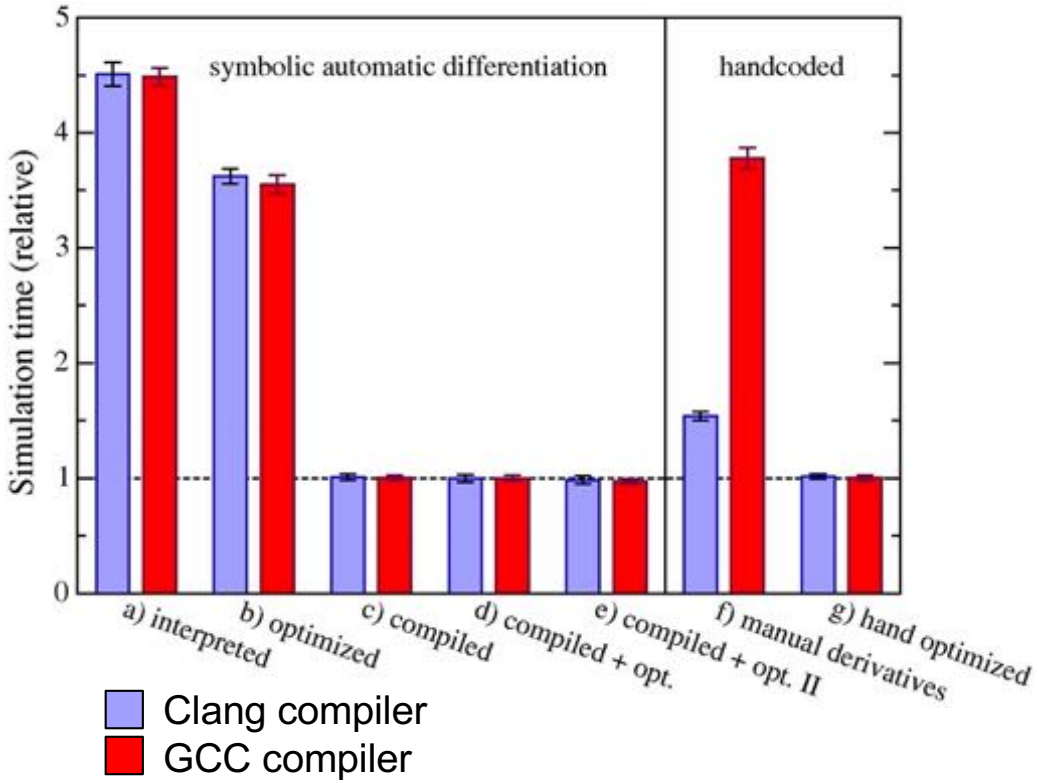
- Based on FunctionParser library
  http://warp.povusers.org/FunctionParser/
  to allow **runtime** specification of mathematical expressions
  - Mathematical expressions become tree data structures connected by operators
  - Differentiation rules are applied recursively starting at the root of the tree
- Automatic differentiation:
  - Eliminates sources of human error
  - Conserves developer time



Schwen, Daniel, et al. Computational Materials Science 132 (2017): 36-45.

The interpreted functions are just as fast as compiled functions

- Just In Time (JIT) compilation speeds up the FParser functions
- ~80ms compile time per function. Results cached.
- There is no reason not to use the interpreted functions with automatic differentiation



Schwen, Daniel, et al. Computational Materials Science 132 (2017): 36-45.

The functions are defined in the input file using natural and straight forward syntax

- The automatic differentiation was developed specifically for phase field
- You can use:
  - Coupled non-linear and auxiliary variables

    ```
    args = ‘cv ci’
    ```

  - Material properties

    ```
    material_property_names = ’W cv_eq’
    ```

  - Constants

    ```
    constant_names       = 'T    kB           Efv cmveq            cbveq‘
    constant_expressions = '1800 8.6173324e-5 3.0 exp(-Efv/(kB*T)) 1‘
    ```

  - The entire feature set of FunctionParser in MOOSE, including
    - Conditionals `if(a>b,B,C)`
    - Inline variables `y:=x^2; sin(2*y)`

Material property dependencies can be included, including derivatives

`material_property_names =`

`F`	A material property called `F` with no declared variable dependencies (i.e. vanishing derivatives)
`F(c,phi)`	A material property called `F` with declared dependence on `c` and `phi`
`d3x:=D[x(a,b),a,a,b]`	The third derivative $\partial^3 x/\partial a^2 \partial b$ of the `a,b`-dependent material property `x`, which will be referred to as `d3x` in the function expression
`dF:=D[F,c]`	Derivative of F w.r.t. `c`. Although the `c`-dependence of `F` is not explicitly declared using the round-bracket-notation, it is implicitly assumed as a derivative w.r.t. `c` is requested

Example 1: Polynomial free energy function

- $f_{loc} = A\left(c - c_A^{eq}\right)^2\left(c - c_B^{eq}\right)^2$

```
[Materials]
  [./f_loc]
    type = DerivativeParsedMaterial
    f_name = f_loc
    args = c
    function = 'A*(c - ca)^2*(c - cb)^2'
    constant_names = 'A ca cb'
    constant_expressions = '5 0.3 0.7'
  [../]
[]
```

Example 2: Ideal solution model free energy function

- $f_{loc} = E_f c + k_b T(c \ln c + (1-c)\ln(1-c))$
- The plog(val,tol) function takes the natural log, unless val < tol and then switches to a Taylor approximation

```
[Materials]
  [./f_loc]
    type = DerivativeParsedMaterial
    f_name = f_loc
    args = 'c T'
    function = 'Ef*c + kb*T*(c*plog(c, tol) + (1 – c)*plog(1 – c, tol))
    constant_names = 'Ef kb tol'
    constant_expressions = '1.2 8.6173303e−5 1.0e-6'
  [../]
[]
```

Example 3: Free energy functional composed of a combination of other free energies

- $f_{loc} = f_A(c)\big(1 - h(\eta)\big) + f_B(c)h(\eta) + \mathrm{W}\,\mathrm{g}(\eta)$
- $f_A(c)$, $f_A(c)$, $h(\eta)$ and $\mathrm{g}(\eta)$ would be created separately with additional DerivativeParsedMaterial blocks

```
[Materials]
  [./f_loc]
    type = DerivativeParsedMaterial
    f_name = f_loc
    args = 'c eta'
    function = 'fA*(1 – h) + fB*h + W*g'
    material_property_names = 'fA(c) fB(c) h(eta) g(eta)'
    constant_variables = 'W'
    constant_expressions = '1.0'
  [../]
[]
```

Example 4: Mobility that is a function of the second derivative of the free energy density

- $M = D / \frac{\partial^2 f_{loc}}{\partial c^2}$
- f_{loc} would be created by another derivative parsed material and D would be a standard material property

```
[Materials]
  [./f_loc]
    type = DerivativeParsedMaterial
    f_name = f_loc
    args = 'c'
    function = 'D/D2f_loc'
    material_property_names = 'D D2f_loc:=D(f_loc, c, c)'
  [../]
[]
```

MOOSE has a number of possible Boundary conditions that can be applied

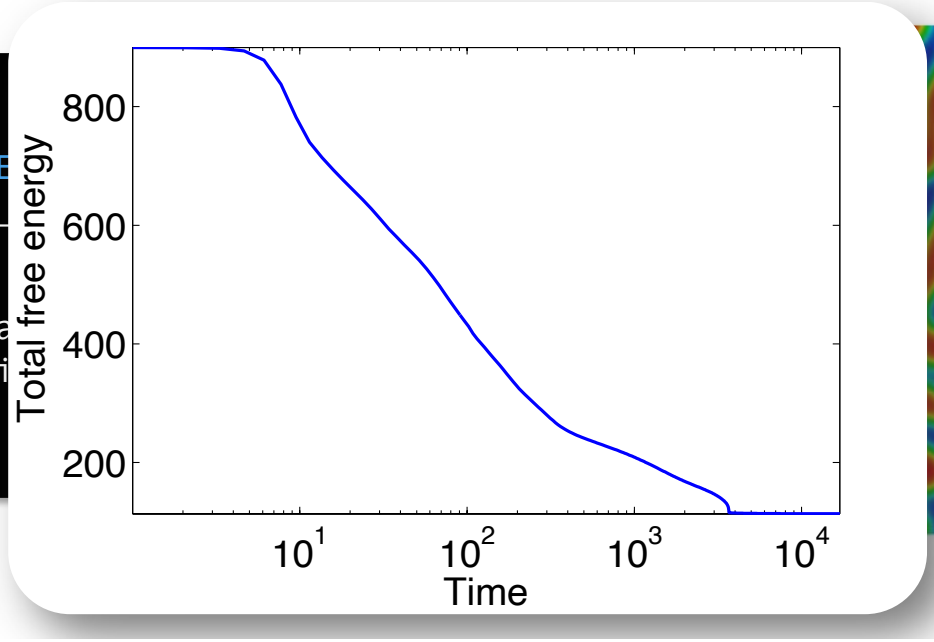
- Some of the existing boundary conditions are:
  - PresetBC – Assigns the value of a variable on a boundary
  - NeumannBC – Assigns the derivative of a variable
  - PeriodicBC – Makes a boundary periodic
- Users can also write their own BCs

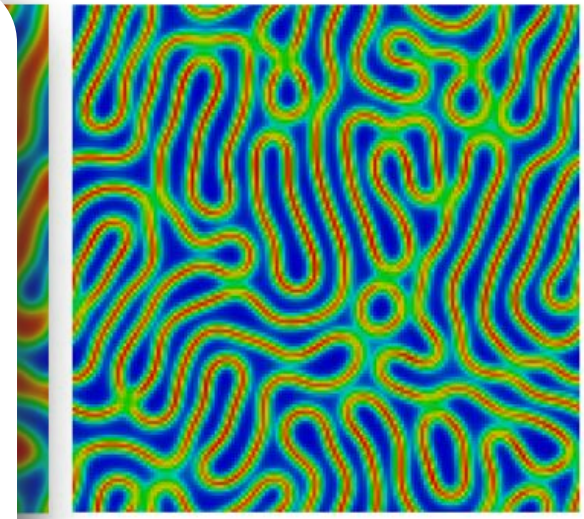
```
[BCs]
  [./Periodic]
    [./all]
      auto_direction = 'x y'
    [../]
  [../]
[]
```

The TotalFreeEnergy *auxkernel adds the gradient energy density to the other terms*

- $f_{total} = f_{loc}(c_i, \eta_j, \ldots, T) + E_d(c_i, \eta_j) + \sum_i \frac{\kappa_i}{2} (\nabla c_i)^2 + \sum_j \frac{\kappa_j}{2} (\nabla \eta_j)^2$
- The total free energy F in the system is computed using the `ElementVariableIntegralPostprocessor`
- **It is always a good idea to check to make sure that the total free energy is going down in your simulation**

```
[AuxKernels]
  [./local_energy]
    type = TotalFreeE
    variable = local_
    f_name = fbulk
    interfacial_vars
    kappa_names = 'ka
    execute_on = 'ini
  [../]
[]
```





Free energy density

The evolving microstructure can be quantified using postprocessors

- FeatureFloodCount – A postprocessor that counts the number of distinct regions in a given variable, defined by a threshold value.
- FeatureVolumeVectorPostprocessor – A vector postprocessor that compiles accurate volumes from the FeatureFloodCount
- FeatureVolumeFraction – A postprocessor that computes the volume fraction defined by the FeatureFloodCount variable

```
[./feature_counter]
  type = FeatureFloodCount
  variable = c
  threshold = 0.5
  compute_var_to_feature_map = true
[../]
```

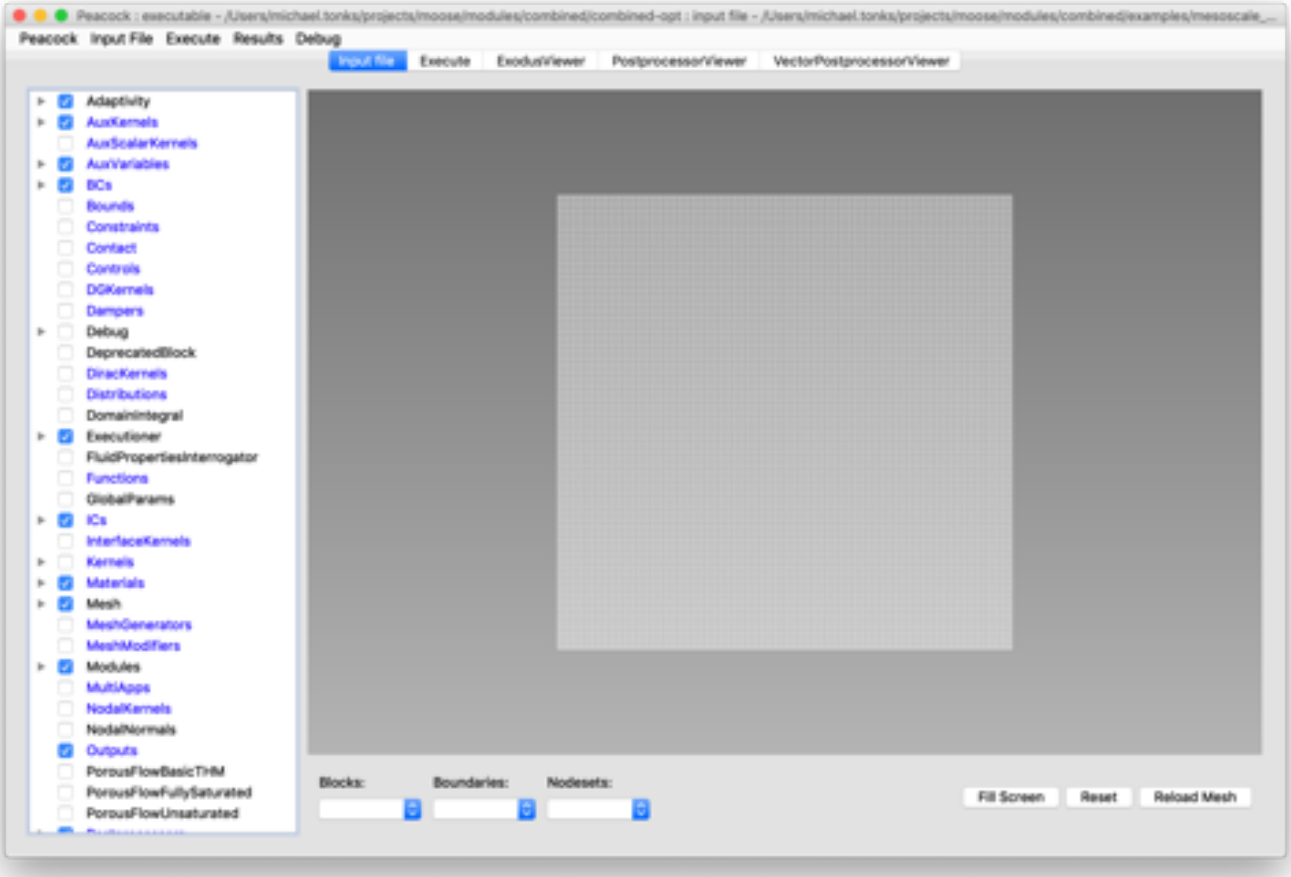
```
[VectorPostprocessors]
  [./feature_volumes]
    type = FeatureVolumeVectorPostprocessor
    flood_counter = feature_counter
  [../]
[]
```

```
[./Volume_Fraction]
  type = FeatureVolumeFraction
  mesh_volume = Volume
  feature_volumes = feature_volumes
[../]
```

```
[./Volume]
  type = VolumePostprocessor
  execute_on = 'INITIAL'
[../]
```

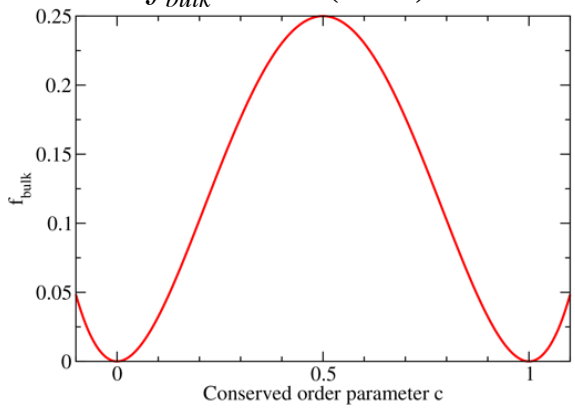
MOOSE comes with a graphic user interface called PEACOCK

- PEACOCK simplifies the creation of input files, execution of MOOSE, and visualization of results

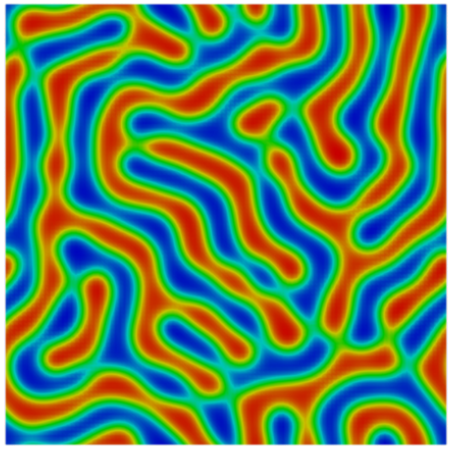


To summarize, the phase field method is a useful tool and MOOSE makes it easy

$$f_{bulk} = 4c^2(1-c)^2$$

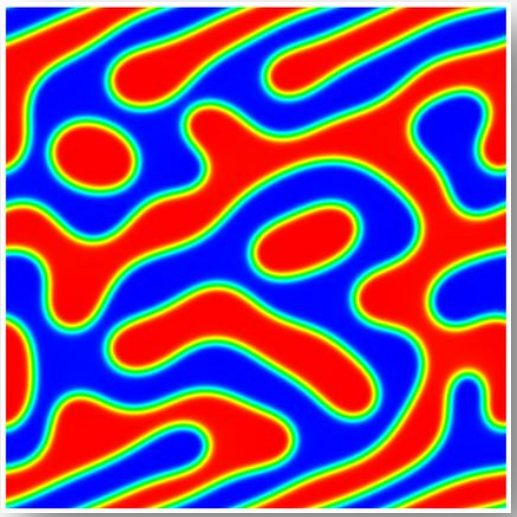



```
[Materials]
  [./f_loc]
    type = DerivativeParsedMaterial
    f_name = f_loc
    args = c
    function = '4.0*c^2*(c - 1)^2'
  [../]
[]
```



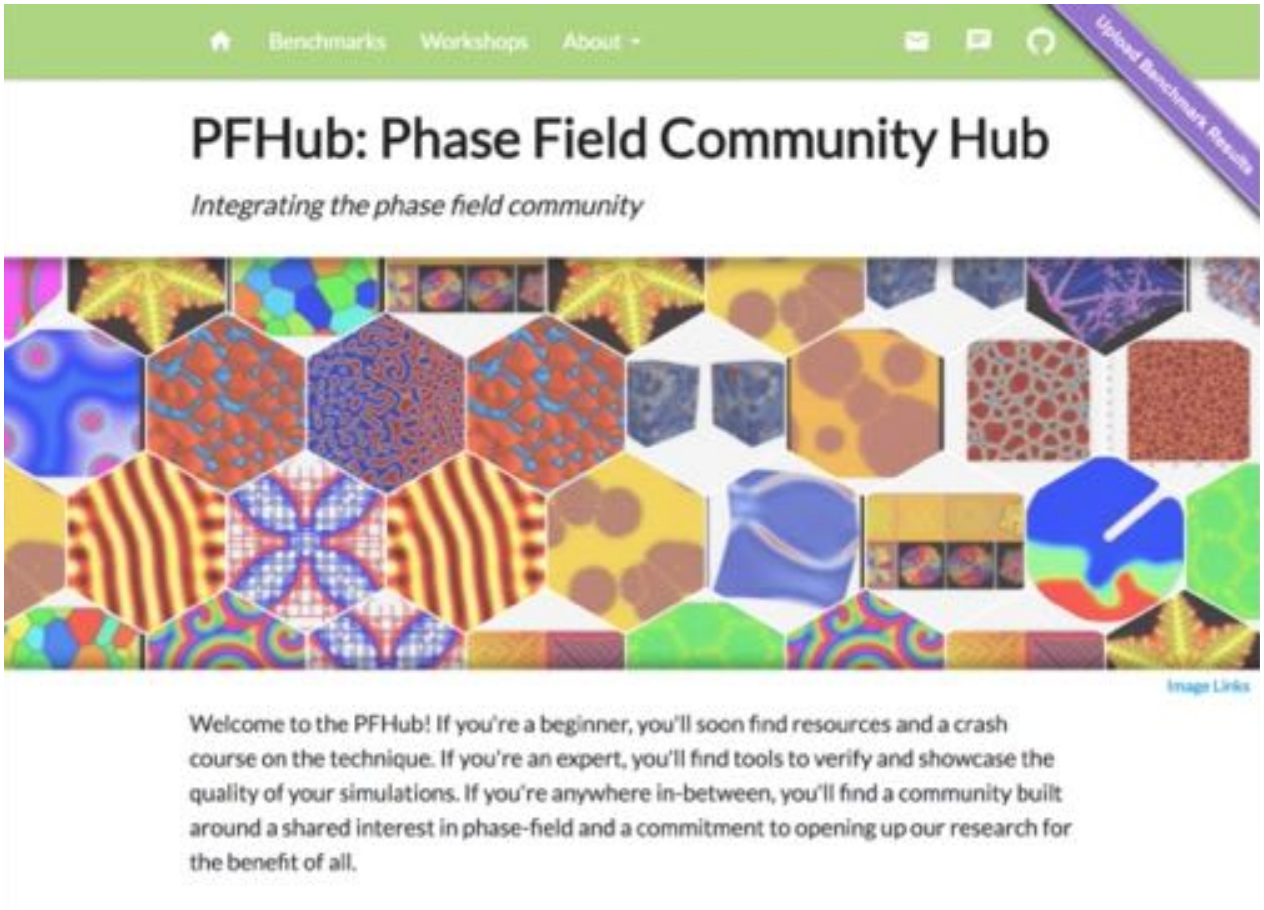
Phase Field Module

First example problem



CHiMaD/NIST Spinodal Decomposition Benchmark Problem

- https://pages.nist.gov/pfhub/benchmarks/benchmark1.ipynb/



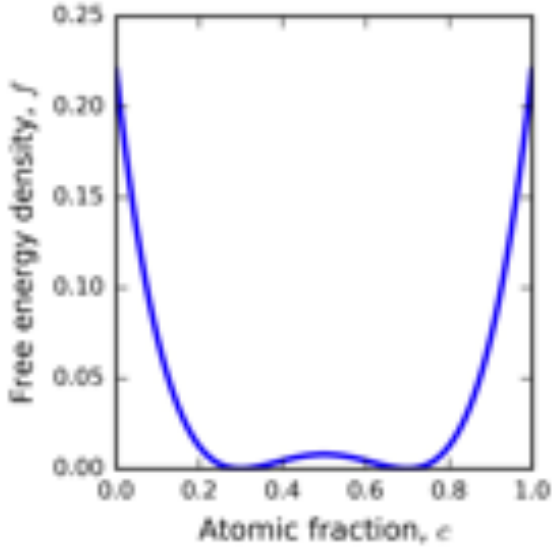
CHiMaD/NIST Spinodal Decomposition Benchmark Problem

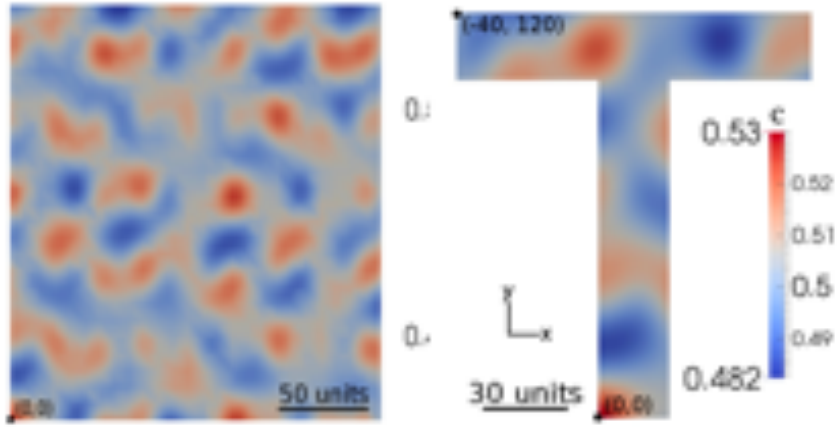
One conserved order parameter

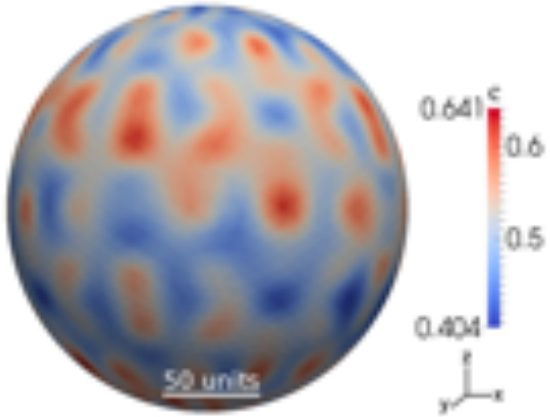
$$F = \int_V \left(f_{chem}(c) + \frac{\kappa}{2} |\nabla c|^2 \right) dV$$

$$f_{chem}(c) = \rho_s (c - c_\alpha)^2 (c - c_\beta)^2$$

$$\frac{\partial c}{\partial t} = \nabla \cdot \left\{ M \nabla \left(\frac{\partial f_{chem}}{\partial c} - \kappa \nabla^2 c \right) \right\}$$







Domains and initial conditions

Input File Blocks – Mesh and ICs

```
[Mesh]
  type = GeneratedMesh
  xmax = 200
  ymax = 200
  nx = 200
  ny = 200
  dim = 2
[]

[ICs]
  [./c_IC]
    type = FunctionIC
    variable = c
    function = '0.5 + 0.01*(cos(0.105*x)*cos(0.11*y)
                + (cos(0.13*x)*cos(0.087*y))^2
                + cos(0.025*x - 0.15*y)*cos(0.07*x - 0.02*y'
  [../]
[]
```

Input File Blocks – Actions and Materials

```
[Modules]
  [./PhaseField]
    [./Conserved]
      [./c]
        free_energy = F
        kappa = kappa
        mobility = M
        solve_type = FORWARD_SPLIT
      [../]
    [../]
  [../]
[]
```

```
[Materials]
  [./constant_props]
    type = GenericConstantMaterial
    prop_names = 'kappa M'
    prop_values = '2 5'
  [../]
  [./F_mat]
    type = DerivativeParsedMaterial
    f_name = F
    args = c
    function = 'A*(c-ca)^2*(c-cb)^2'
    constant_names = 'A ca cb'
    constant_expressions = '5 0.3 0.7'
  [../]
[]
```

Input File Blocks – AuxKernels and BCs

```
[AuxVariables]
  [./Fdens]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]

[AuxKernels]
  [./Fdens]
    type = TotalFreeEnergy
    variable = Fdens
    interfacial_vars = c
    kappa_names = kappa
    f_name = F
    execute_on = TIMESTEP_END
  [../]
[]

[BCs]
  [./Periodic]
    [./All]
      auto_direction = 'x y'
    [../]
  [../]
[]
```

Input File Blocks – Preconditioning, Executioner and Outputs

```
[Executioner]
  type = Transient
  solve_type = NEWTON
  scheme = BDF2
  petsc_options_iname = '-pc_type
                         -sub_pc_type
                         -pc_asm_overlap'
  petsc_options_value = 'asm lu 2'
  nl_max_its = 12
  l_max_its = 40
  l_tol = 1e-5
  nl_rel_tol = 1e-6
  end_time = 1e6

  [./TimeStepper]
    type = IterationAdaptiveDT
    dt = 0.05
    growth_factor = 1.1
    cutback_factor = 0.8
    optimal_iterations = 6
  [../]
[]
```

```
[Preconditioning]
  [./SMP]
    type = SMP
    full = true
  [../]
[]

[Outputs]
  exodus = true
  csv = true
  perf_graph = true
[]
```