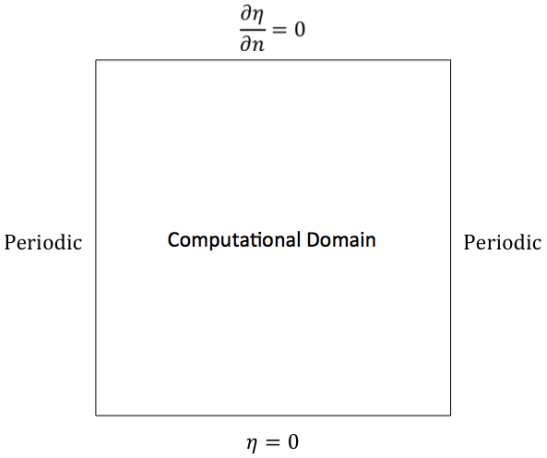
PRISMS-PF Training Exercises:

Here is a set of exercises to familiarize yourself with PRISMS-PF. Most users find that these problems take several hours to complete in a training environment where questions can be answered in real time. The problems are approximately in ascending order of difficulty. We recommend copying and renaming the example application directories before making modifications so that you still have the original versions to refer to. Delete the file "CMakeCache.txt" in the newly created directory.

1. **Boundary Conditions I:**
   Changing boundary conditions for the Allen-Cahn example problem
   a. Change the BCs in the Allen-Cahn application to zero flux (the natural BC) on the top boundary, eta = 0 on the bottom boundary (a Dirichlet BC), and periodic on the two side boundaries (see diagram below)



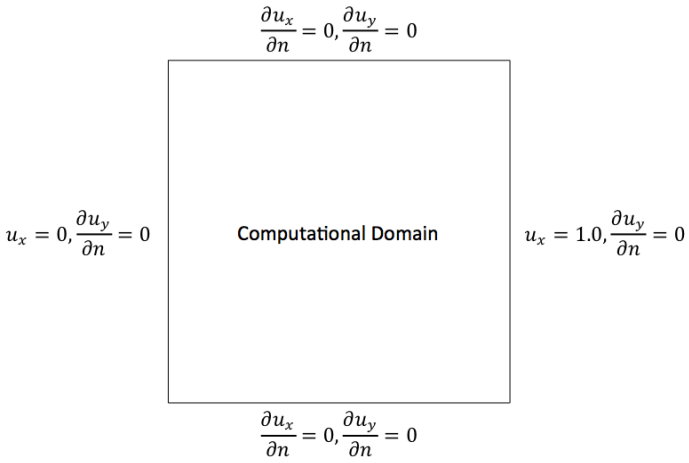
2. **Adaptivity:**
   Add adaptivity to the Allen-Cahn example problem
   a. Copy the adaptivity section of parameters_adaptive.in from the cahnHilliard application into the Allen-Cahn application, changing the variable that determines the mesh adaptivity from "c" to "n". See how much you can decrease the run time by using adaptivity, adjusting the parameters so that the solution doesn't change significantly. (Note that the initial refine factor should be between the max and min levels of refinement.)
   b. Repeat with the minimum mesh size decreased by a factor of 2 in each direction. For stability purposes, the time step will need to be decreased by a factor of 4.

3. **Postprocessing:**
   Use the PRISMS-PF postprocessor to verify that the integral of a field is conserved. (Note that this is a bit of a degenerate postprocessed field, typically the postprocessed fields would be different than the "primary" fields listed in equations.cc. We're adding one of the primary fields as a postprocessed field because only postprocessed fields can be integrated.)
   a. Add a second postprocessing variable in the cahnHilliard application for the concentration
   b. Run a simulation and verify that the integral of the concentration is constant by examining the output in the file "integratedFields.txt".

4. **Initial Conditions and Computational Domain:**
   Adding a third particle to the coupled Allen-Cahn/Cahn-Hilliard application
   a. Increase the size of the computational domain by 50% in each direction, using the same mesh size as in the original calculation (i.e. you will need to increase the total number of elements from 192 in each direction to 288).
   b. Add a third particle to the initial condition in a location of your choice that doesn't overlap with either of the other two particles. (Note: if the particles do overlap, the simulation will likely crash. However, you should be able to open the initial output in VisIt.)

5. **Model constants:**
   Add more model constants to an application
   a. In the allenCahn application, the derivative of the free energy is hardcoded in equations.cc through the variable "fnV". The corresponding free energy expression is:
   $$f_{chem} = \eta^4 - 2\eta^3 + \eta^2$$
   b. Make the necessary modifications to parameters.in, equations.cc, and customPDE.h to turn this into a generic 4th order polynomial with coefficients set in parameters.in as more "Model constants".
   c. Also make the necessary changes in postprocess.cc so that the free energy expression is correct for outputted total free energy
   d. Note how the solution changes when the homogenous free energy is changed to:
   $$f_{chem} = 4\eta^4 - 8\eta^3 + 4\eta^2$$
   by changing the constants in parameters.in

6. **Boundary Conditions II:**
   Strained precipitate evolution
   a. Run the un-modified precipitateEvolution example and move the output files into a new directory named “no_strain”.
   b. Change the boundary conditions for the precipitate evolution application so that the x=40 boundary is displaced by +1 unit, and that the boundary condition on the x component of the displacement along the y=0 and y=40 boundaries are zero derivative (see diagram below). Move the output files to a new directory named “tension” and compare the results to the unstrained results.
   c. Repeat, but with the x=40 boundary displaced by -0.2 units, moving the output files to a new directory named “compression”.
   d. Change the boundary condition on the x=40 to a Dirichlet BC that increases from 0 to +1 over the course of the simulation (Hints: use a NON_UNIFORM_DIRICHLET boundary condition, and the current time can be accessed in non-uniform Dirichlet BC function via the variable “time”)



7. **Governing Equations I:**
   Add a barrier term to the Precipitate Evolution application
   a. Move two precipitates with different order parameters closer together and observe what happens when they touch (space them far apart enough that it takes a few thousand time steps for this to happen).
   b. Add a penalty term for overlapping structural order parameters. The barrier in the free energy is:

$$f_{barrier} = n_1^2 n_2^2 + n_1^2 n_3^2 + n_2^2 n_3^2 + n_1^2 n_2^2 n_3^2$$

      In each Allen-Cahn equation, this adds a term to the "eq_ni" term equal to $-\Delta t\ M\ \frac{\partial f_{barrier}}{\partial n_i}$, where "i" is the index of the structural order parameter.
   c. Add the barrier term to the calculation of the total free energy, note how much it shifts the energy.

8. **Grain Remapping**
   Change the number of order parameters available for remapping
   a. Run the grainGrowth app and move the result files into a directory named 'six_ops'.
   b. Modify the input file so that the variable 'n5' is not one of the variables available for remapping. All of the initial grains are placed on order parameters 'n0' through 'n4', so 'n5' will remain empty throughout the simulation. If you wanted to, you could fully remove 'n5' from the app, but that is unnecessary for this exercise. Rerun the simulation and move the results into a directory named 'five_ops'.
   c. Visualize the results of both simulations. Are they the different? If so, how?
   d. Reduce the number of time steps between grain reassignments until you get the same result as with all six order parameters.

9. **Governing Equations II:**
   Create a new application that simulates the growth of particle with a kinetically determined faceted shape. Similar models have been used for simulating selective area epitaxy and electrodeposition.
   a. Create a new application by copying the "cahnHilliard" application and renaming it "kineticWulffShape". Delete the "CMakeCache.txt" file.
   b. Change the initial condition to a circle with radius of 12 units in the center of the domain. Run a (short) simulation to check that the initial condition is correct.
   c. Add a source term to the "eq_c" term: 0.05*McV*userInputs.dtValue* std::max(c*(1.0-c),constV(0.0)). This source term will add mass at the interface, causing the initial particle to grow. Run a simulation to make sure that the particle is growing.

d. Multiply the source term by an anisotropy coefficient, γ, where:

$$\gamma = 1 + 0.8 * \left[4 * \left(\left(\frac{\frac{\partial c}{\partial x}}{|\nabla c| + 1.0\times10^{-10}} \right)^4 + \left(\frac{\frac{\partial c}{\partial y}}{|\nabla c| + 1.0\times10^{-10}} \right)^4 \right) - 3 \right]$$

Run the simulation to see how the orientation-dependent source term causes the morphology to change.

10. Computational Domain II (Warning: Requires some C++ knowledge)**:**
Override the core PRISMS-PF function that creates the mesh to create a circular domain

a. Add the following member function to customPDE for the allenCahn application (add it to the bottom of customPDE.h and add the function declaration to the "Methods specific to this subclass" section of the customPDE class declaration):

```
template <int dim, int degree>
void
customPDE<dim,degree>::makeTriangulation(parallel::distribu
ted::Triangulation<dim> & tria) const{

    GridGenerator::hyper_ball (tria,
Point<dim>(userInputs.domain_size[0],userInputs.domain_size
[1]/2.0), userInputs.domain_size[1]/2.0);

       // Attach a spherical manifold to the semicircular
part of the domain so that it gets refined with rounded
edges
    static const SphericalManifold<dim>
boundary(Point<dim>(userInputs.domain_size[0],userInputs.do
main_size[1]/2.0));
    tria.set_manifold(0,boundary);

}
```

This will override the "makeTriangulation" function in the MatrixFreePDE class that can only create rectangular meshes. Instead, this function uses the "hyper_ball" mesh generator from deal.II that creates circular or spherical domains. Other mesh shapes that deal.II can generate are given here: https://www.dealii.org/8.5.0/doxygen/deal.II/namespaceGridGenerator.html.

b. Reduce the refine factor in parameters.in to 6 (the coarsest mesh for a circular domain is still partially refined, reducing the amount we have to refine it further)