Scientific Computation of Two–Phase Ferrofluid Flows
AMSC 664 Proposal

Gareth Johnson
Faculty Adviser: Ricardo Nochetto

February 8, 2019

1 Semester Plan

The main goal of this project remains unchanged from last semester. I intend to implement a two–phase ferrofluid flow solver using the algorithm presented in [5], which is duplicated below as it will be referred throughout this proposal.

Algorithm 1. For given smooth initial data $\left\{\Theta^0, \mathbf{M}^0, \mathbf{U}^0\right\}$, compute $\left\{\Theta^k, \Psi^k, \mathbf{M}^k, \Phi^k, \mathbf{U}^k, P^k\right\} \in \mathbb{G}_h \times \mathbb{Y}_h \times \mathbb{M}_h \times \mathbb{X}_h \times \mathbb{U}_h \times \mathbb{P}_h$ for every $k \in \{1, ..., K\}$ that solves

$$\left(\frac{\delta\Theta^k}{\tau}, \Lambda\right) - (\mathbf{U}^k\Theta^{k-1}, \nabla\Lambda) - \gamma(\nabla\Psi^k, \nabla\Lambda) = 0, \tag{1a}$$

$$(\Psi^k, \Upsilon) + \epsilon(\nabla\Theta^k, \nabla\Upsilon) + \frac{1}{\epsilon}(f(\Theta^{k-1}), \Upsilon) + \frac{1}{\eta}(\delta\Theta^k, \Upsilon) = 0, \tag{1b}$$

$$\left(\frac{\delta\mathbf{M}^k}{\tau}, \mathbf{Z}\right) - \mathcal{B}_h^m(\mathbf{U}^k, \mathbf{Z}, \mathbf{M}^k) + \frac{1}{\mathscr{T}}(\mathbf{M}^k, \mathbf{Z}) = \frac{1}{\mathscr{T}}(\varkappa_\theta\mathbf{H}^k, \mathbf{Z}), \tag{1c}$$

$$(\nabla\Phi^k, \nabla X) = (\mathbf{h}_a^k - \mathbf{M}^k, \nabla X), \tag{1d}$$

$$\begin{aligned}\left(\frac{\delta\mathbf{U}^k}{\tau}, \mathbf{V}\right) + \mathcal{B}_h(\mathbf{U}^{k-1}, \mathbf{U}^k, \mathbf{V}) + (\nu_\theta\mathbf{T}(\mathbf{U}^k), \mathbf{T}(\mathbf{V})) - (P^k, \mathrm{div}\mathbf{V}) &= \mu_0\mathcal{B}_h^m(\mathbf{V}, \mathbf{H}^k, \mathbf{M}^k) \\ &+ \frac{\lambda}{\epsilon}(\Theta^{k-1}\nabla\Psi^k, \mathbf{V}),\end{aligned} \tag{1e}$$

$$(Q, \mathrm{div}\mathbf{U}^k) = 0, \tag{1f}$$

for all $\{\Lambda, \Upsilon, \mathbf{Z}, X, \mathbf{V}, Q\} \in \mathbb{G}_h \times \mathbb{Y}_h \times \mathbb{M}_h \times \mathbb{X}_h \times \mathbb{U}_h \times \mathbb{P}_h$, where $\mathbf{H}^k = \nabla\Phi^k$ and $\eta \leq \left(\max_\theta f'(\theta)\right)^{-1}$.

The implementation will be written in C++ and will utilize the deal.II library [1, 2]. The code will be written to take advantage of deal.II's dimensionless programming design allowing the transition between 2d to 3d without having to rewrite large portions of the code base. However, due to computational resources available to the project we will only focus on the 2d case for our testing purposes.

As pointed out in last semesters final review, the project is behind the timeline that was given in the original project proposal. Last semester I developed functionality to generate the applied magnetizing field from a given set of magnetic dipoles and to generate the initial mesh. Both pieces of functionality were united tested, though admittedly not described in full detail, and thus will be included in the unit testing section below. In order to better track the progress of my project this semester I have broken the remaining work into the following four main tasks.

- The first task will be to develop code to solve the Cahn–Hilliard system (1a)–(1b). In addition to solving the system, this task will include functionality for adaptive mesh refinement/coarsening. This is because we are using the variable Θ in the definition of the Kelly error estimator [4]

$$\eta_T^2 = h_T \int_{\partial T} \left| \left[\!\left[\frac{\partial\Theta}{\partial\eta} \right]\!\right] \right|^2 dS \quad \forall T \in \mathcal{T}_h. \tag{2}$$

Specifically, this error estimator will be used to determine which elements will be coarsened or refined. deal.II provides functionality to coarsen a percentage of elements with the smallest error and refine a percentage of elements with the largest error. After marking elements for refinement/coarsening another deal.II method will be used to transfer the solution from the current mesh to the new refined/coarsened mesh. Note, that while each of the six variables will be transfered from the current mesh to the coarsened/refined mesh, the decision of which elements to coarsen/refine is solely based on the variable Θ.

- The second task will be to develop code to solve the Navier–Stokes system (1e)–(1f).
- The third task will be to develop code to solve the magnetization system (1c)–(1d).
- The final task will to solve the overall scheme using a Picard–like iteration.

I believe that getting the above four tasks finished in this semester will be a challenge, as each of the three solvers are drastically different in addition to getting them all to work together. Thus my plan is to progress through the semester focusing on one task at a time until is complete and passing the unit test.

2 Code Units

The above tasks can be summarized into the following set of code units:

- The Cahn–Hilliard solver will be a single class with the following methods:
  - A constructor which is responsible for reading in the initial condition.
  - A subroutine for setting up the system. This will be comprised of distributing the degrees of freedom and computing the matrices resulting from the FEM formulation.
  - A subroutine which will solve the system at the current timestep. This system will be solved using GMRES preconditioned with algebraic multigrid.
  - A subroutine to coarsed/refine the mesh. Using the procedure described elements will be marked for coarsening/refinement. The coarsened/refined mesh will then be available to the other classes to that they can transfer their variables onto the new mesh.
- The Navier–Stokes solver will be a single class with the following methods:
  - A constructor which is responsible for reading in the initial condition.
  - A subroutine for setting up the system. This will be comprised of distributing the degrees of freedom and computing the matrices resulting from the FEM formulation.
  - A subroutine which will solve the system at the current timestep. As the system is a non–symmetric saddle point problem, GMRES with a block preconditioner for the saddle point system will be used. Specifically, I plan on exploring preconditioners presented in Elman's book [3].
  - A subroutine to transfer the solution from the current mesh to the new refined/coarsened mesh.
- The magnetization solver will be a single class with the following methods:
  - A constructor which is responsible for reading in the initial condition.

  - A subroutine for setting up the system. This will be comprised of distributing the degrees of freedom and computing the matrices resulting from the FEM formulation.
  - A subroutine which will solve the system at the current timestep. The system will be solved using both BiCGstab and CG preconditioned with algebraic multigrid.
  - A subroutine to transfer the solution from the current mesh to the new refined/coarsened mesh.

- The Picard–like iteration will be a subroutine which solves the entire system at a single timestep. It will repeatedly solve the above three subsystems, lagging the velocity variable, until the velocity has reached a fixed–point.

3 Unit Testing

The input mesh generation was unit tested by visually inspecting the generated mesh. This is verifiable as the code takes in 2 numbers which describe how many elements to have in the vertical and horizontal direction. The generation of the applied magnetizing field was unit tested by inputting a known set of magnetic dipoles and computing the applied magnetizing field at a few points on a mesh. This was compared to the analytically computed value and output a message only if the error was larger than $1E-6$.

The first three tasks will be unit tested using the technique of generated solutions. Specifically, for each system a forcing function will be added to make the system hold for a predetermined analytic solution. Then I will compute and export the L^2 and L^∞ errors to ensure that each solution is converging to a threshold of $1E-6$. In order to actually run the test a modified version of the above classes will be created. The modification is required in order to add in the forcing term in the subroutine responsible for setting up each system. Finally, note that I am not planning to individually test the "sub items" that are listed above as these subroutines are primarily comprised of methods from the deal.II library. Thus the technique of generated solutions will test whether or not have used those methods correctly in order to solve the system. Below are descriptions of the analytical solutions that will be used for each system

- Cahn–Hilliard equation: The analytical solution for the phase variable will consist of the following three portions. The left half of the mesh will have the phase being -1, the right half of the mesh will have the phase being 1, and the center region of length $\epsilon = .001$ will have the profile $\tanh(\frac{x}{\sqrt{2\epsilon}})$.

- Navier–Stokes equation: The analytical solution for the velocity will be a radial constant flow. The flow will have speed 0 at the center of the mesh and the flow speed will increase linearly as you move out from the center of the mesh.

- Magnetization equation: The analytical solution for the magnetization will be a linear transport in the positive vertical direction with a speed of $v = .5$.

The final task will be unit tested by comparison to visual results presented in [5]. In the paper, three experiments were run which reproduced the Rosenzweig instability with a uniform magnetic field, and the so called "ferrofluid hedgehog" for a non-uniform magnetic field both with and without the inclusion of the de-magnetizing field. By running the full code with the three different configurations given in the paper, I will be able to visually compare my results with the results given in the paper.

References

[1] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler, *Algorithms and data structures for massively parallel generic adaptive finite element codes*, ACM Trans. Math. Softw., 38 (2012), pp. 14:1–14:28.

[2] W. Bangerth, T. Heister, and G. Kanschat, *Deal.ii differential equations analysis library, technical reference.*

[3] H. Elman, D. Silvester, and A. Wathen, *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics*, Oxford University Press, 2005.

[4] D. W. Kelly, J. P. De S. R. Gago, O. C. Zienkiewicz, and I. Babuska, *A posteriori error analysis and adaptive processes in the finite element method: Part ierror analysis*, International Journal for Numerical Methods in Engineering, 19, pp. 1593–1619.

[5] R. H. Nochetto, A. J. Saldago, and I. Tomas, *A diffuse interface model for two-phase ferrofluid flows*, Computer Methods in Applied Mechanics and Engineering, 309 (2016), pp. 497–531.