

ACSE9 - IRP Project Plan

Keer Mei

June 2019

1 Introduction

The goal of this project is to develop a python framework using the Firedrake API for solving problems involving coupled partial differential equations (PDEs). Currently, Firedrake users must have some degree of familiarity with its functionality as well as technical knowledge of the finite element method. A python framework provides clarity and ease of use for users who want to adopt the finite element method by allowing them to issue high level instructions for setting up and solving their own coupled problems.

Inevitably, many different coupled problems will share the same or similar underlying PDEs depending on the governing physical domain. In this regard, the Firedrake framework can also provide developers with a flexible basis to implement their own coupled problems using the already developed foundation classes.

Included within the scope of this project will be demonstrations of how the Firedrake framework is used to solve several different coupled problems.

2 Background

Firedrake employs the Unified Form Language (UFL) which is a domain-specific language that contains built-in support for automatic differentiation and producing variational forms of partial differential equations (PDEs). This allows the user to generate code that resembles closely to mathematical equations.

Although the existing Firedrake API provides an intuitive interface for users to deploy the individual steps of the finite element method, it is not a trivial process. The knowledge to build a mesh, specify initial/boundary conditions, create multiple and/or mixed function spaces, specify solver types, as well as a multitude of other considerations makes using Firedrake as a research tool for coupled PDEs complicated. A framework will simplify this

process by giving users PDE classes with inter-dependencies without needing to worry about how each individual functionality comply with others. An example of a similar framework has been implemented in FEniCS to solve turbulent flow models [1].

2.1 Flow coupled Reactions

One example of a coupled problem is the 2 dimensional incompressible channel flow through a cylinder with a first order chemical reaction [2]. In this problem, we must first solve the 2-D Navier-Stokes (N-S) Equation, given by:

$$\frac{\partial u}{\partial t} + u \cdot \nabla u = \frac{1}{\rho} \nabla p + \mu \nabla^2 u \quad (1)$$

$$\nabla \cdot u = 0 \quad (2)$$

The N-S equation will have to be solved on a finite domain, with a specified mesh size:

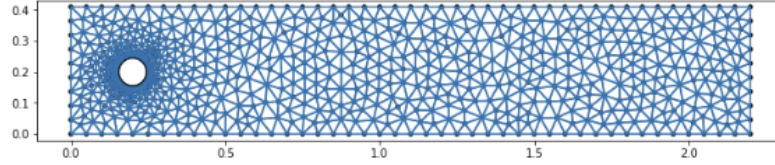


Figure 1: Channel and cylinder mesh

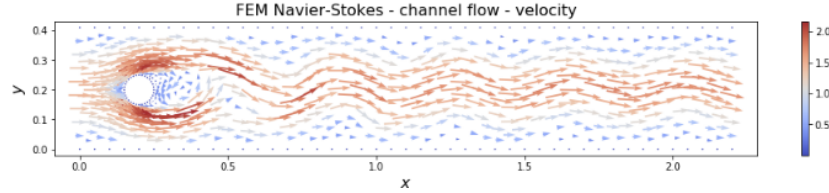


Figure 2: Velocity profile of flow through cylinder

Next, using the velocity u obtained from the solution to the N-S Equation, we can solve for the concentrations of the different chemical components using their mass balance:

$$\frac{\partial c_1}{\partial t} + u \cdot \nabla c_1 - \nabla \cdot (D \nabla c_1) = f_1 - K c_1 c_2 \quad (3)$$

$$\frac{\partial c_2}{\partial t} + u \cdot \nabla c_2 - \nabla \cdot (D \nabla c_2) = f_2 - K c_1 c_2 \quad (4)$$

$$\frac{\partial c_3}{\partial t} + u \cdot \nabla c_3 - \nabla \cdot (D \nabla c_3) = f_3 + K c_1 c_2 - K c_3 \quad (5)$$

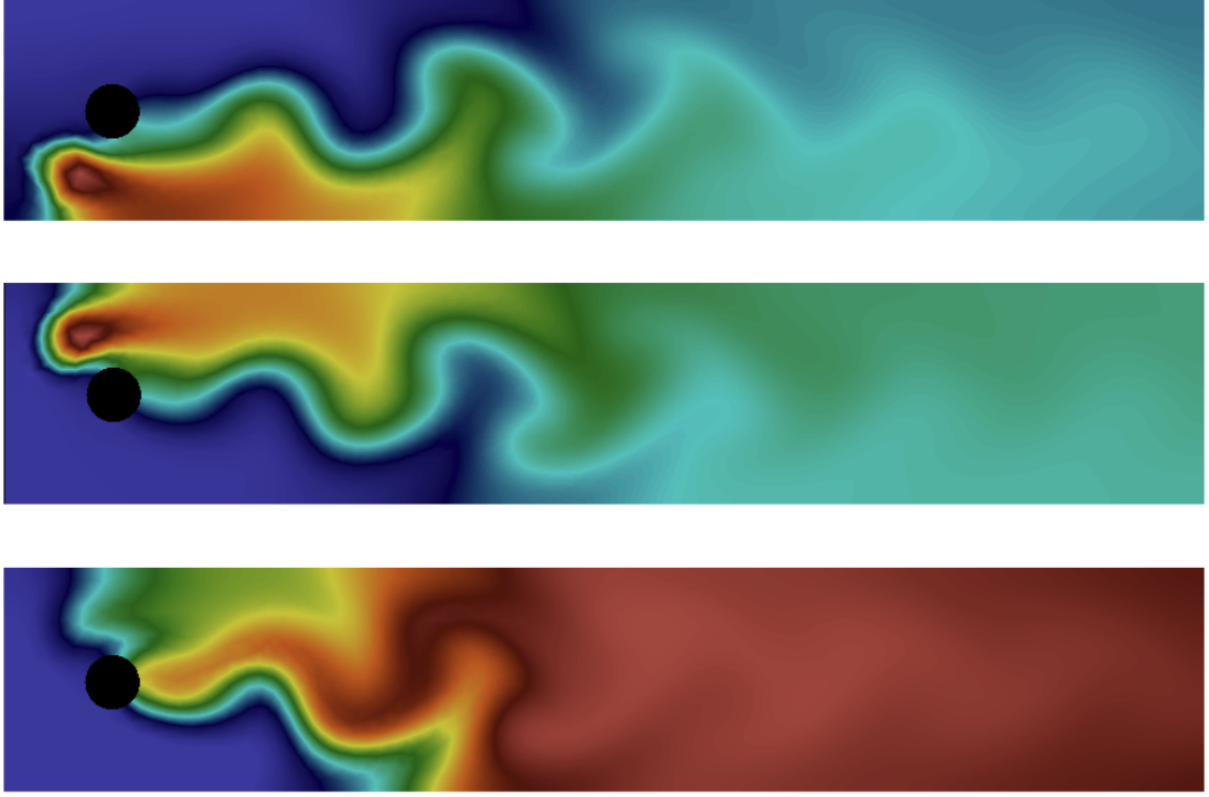


Figure 3: Reactants concentration profiles

2.2 Other coupled problems

Another example of a coupled PDE problem is the modelling of turbulent flow. In research, there exists multiple numbers of different turbulence models such as the Reynolds-averaged Navier-Stokes (RANS), Large Eddy Simulations (LES), and Direct Numerical Simulations (DNS) [5]. Each of these models provide their own advantages and disadvantages and may not be suitable for all scenarios. In the RANS model for example [1], a modified version of the N-S equation must be solved:

$$\frac{\partial U}{\partial t} + U \cdot \nabla U = \frac{1}{\rho} \nabla p + \mu \nabla^2 U \quad (6)$$

$$\nabla \cdot U = 0 \quad (7)$$

$$U = u + u' \quad (8)$$

where U is an average velocity separated into mean(u) and fluctuating(u') components. The difficulty lies in defining the transport models that describe the mean and fluctuating components.

2.3 Existing challenges

As mentioned, the existing Firedrake API equipped with UFL is well suited for converting mathematical expressions into code. However, solving PDEs using Firedrake is still a complicated process that involves multiple steps, which creates opportunities for programming errors. The following excerpt demonstrates the basic necessities for solving the N-S reactions coupled problem described in section 2.1:

```
import firedrake as fd
# physical constants
nu = fd.Constant(0.001)
# time step
dt = 0.001
# define a firedrake constant equal to dt so that variation forms
# not regenerated if we change the time step
k = fd.Constant(dt)

# diffusion rate
eps = fd.Constant(0.01)
# reaction rate
K = fd.Constant(10.0)

# create a mesh
mesh = fd.Mesh("flow_past_cylinder.msh")

# create Function spaces
V = fd.VectorFunctionSpace(mesh, "CG", 2)
Q = fd.FunctionSpace(mesh, "CG", 1)

u = fd.TrialFunction(V)
v = fd.TestFunction(V)
p = fd.TrialFunction(Q)
q = fd.TestFunction(Q)
...

# creating coupled variables and equations from the reaction formulas
P1 = fd.FiniteElement("CG", 'triangle', 1)
element = fd.MixedElement([P1, P1, P1])
H = fd.FunctionSpace(mesh, element)

v_1, v_2, v_3 = fd.TestFunctions(H)
c = fd.Function(H)
c_n = fd.Function(H)
...
```

```

n = fd.FacetNormal(mesh)
f = fd.Constant((0.0, 0.0))
u_mid = 0.5*(u_n + u)

x, y = fd.SpatialCoordinate(mesh)

# create source terms
f_1 = fd.conditional(pow(x-0.1, 2)+pow(y-0.1,2)<0.05*0.05, 0.1, 0)
f_2 = fd.conditional(pow(x-0.1, 2)+pow(y-0.3,2)<0.05*0.05, 0.1, 0)
f_3 = fd.Constant(0.0)

# Define boundary conditions
bcu = [fd.DirichletBC(V, fd.Constant((0,0)), (1, 4)),
      ...]

# Define variational forms
F1 = fd.dot((u - u_n)/k, v) * fd.dx \
      + fd.dot(fd.dot(u_n, fd.nabla_grad(u_n)), v) * fd.dx \
      + fd.inner(sigma(u_mid, p_n), fd.sym(fd.nabla_grad(v))) * fd.dx \
      + ...

a1, L1 = fd.system(F1)

a2 = fd.dot(fd.nabla_grad(p), fd.nabla_grad(q)) * fd.dx
L2 = fd.dot(fd.nabla_grad(p_n), fd.nabla_grad(q)) * fd.dx \
      - (1/k) * fd.div(u_n)*q*fd.dx

a3 = fd.dot(u, v) * fd.dx
L3 = fd.dot(u_n, v) * fd.dx \
      - k * fd.dot(fd.nabla_grad(p_n - p_n), v) * fd.dx

F4 = ((c_1 - c_n1) / k)*v_1*fd.dx \
      + ((c_2 - c_n2) / k)*v_2*fd.dx \
      + ((c_3 - c_n3) / k)*v_3*fd.dx
F4 += ...

# Define problems classes
prob1 = fd.LinearVariationalProblem(a1, L1, ...)
prob2 = fd.LinearVariationalProblem(a2, L2, ...)
prob3 = fd.LinearVariationalProblem(a3, L3, ...)
prob4 = fd.NonlinearVariationalProblem(F4, c, ...)

# Define solvers classes
solve1 = fd.LinearVariationalSolver(prob1, solver_parameters={...})

```

```

solve2 = fd.LinearVariationalSolver(prob2, solver_parameters={...})
solve3 = fd.LinearVariationalSolver(prob3, solver_parameters={...})
solve4 = fd.NonlinearVariationalSolver(prob4, solver_parameters={...})

# Time loop
t = 0.0
t_end = 5.
while t < t_end :
    t += dt
    solve1.solve()
    ...

```

From the excerpt above, several difficulties with using the current Firedrake API become obvious:

1. First, Firedrake users must understand the basics of how to set up function spaces, boundary conditions, conditional source terms, etc...
2. Next, although it is not mentioned, the variational forms written in this example uses a pressure projection method, AKA Chorin projection, to solve for velocity. This is knowledge that the user's must be aware of before even attempting to use Firedrake.
3. What are the appropriate solver parameters that a user should specify? How does an user know if the time step size or the mesh grid size is appropriate for their problem?
4. Finally, the current implementation is simply too long and onerous to solve for larger, more coupled problems.

The end goal is to use the Firedrake framework to simplify the problem and address the challenges above. Similar to the FEniCS framework [1], the user should only be required to specify high level programming instructions in a few short lines of code. Below is an example of how the FEniCS framework solves the N-S equation:

```

import cbc.rans.nsproblems as nsproblems
import cbc.rans.nssolvers as nssolvers
class MyProblem(nsproblems.NSProblem):
    ...
    nsproblems.parameters.update(Nx=10, Ny=10, Re=100)
    problem = MyProblem(nsproblems.parameters)
    nssolvers.parameters = recursive_update(
        nssolvers.parameters, degree=dict(velocity=2,
        pressure=1), scheme_number=dict(velocity=1))
    solver = nssolvers.NSCoupled(problem, nssolvers.parameters)
    solver.setup()
    problem.solve(max_iter=20, max_err=1E-4)
    plot(solver.u_); plot(solver.p_)

```

3 Goals and Milestones

The following are high level targets that have been set by myself based on discussions between myself and my supervisor. I am confident that meeting these deadlines will ensure a smooth progression towards the final deliverable. *Disclaimer:* this is an initial blueprint and certain milestones may not be completed in the planned order or to the same specification.

3.1 June 28, 2019 Milestone

The goal set for this initial period was to accomplish the following:

1. Familiarize myself with writing variational forms of PDEs by reviewing lecture notes. Gain a better understanding of the Firedrake API by creating my own python notebook and attempt to duplicate tutorial problems from Lecture and the Firedrake website. **(Done)**
2. Conduct literature review to collect a variety of coupled PDE problems (RANs turbulence, radionuclide spread in aquatic ecosystems [6], oil spill models [4], reaction models, etc.). Determine potential demonstration problems to be included and solved within the Firedrake framework. **(Ongoing)**

3.2 July 19, 2019 Milestone

By the end of June, I should have acquainted myself enough with the Firedrake API to begin programming a framework. The goal during this period will be to focus on producing the outline of the Firedrake framework.

1. Produce a simple N-S problem and solver class that resembles the examples developed within the FEniCS framework[1]. Once this is accomplished, expand the dependencies and build another class that is coupled with the N-S problem and solver (e.g. either a RANS class or a reactions class). **(Not started)**
2. Finalize literature review and determine the direction of the demonstration problems to be solved - either a complicated coupled problem that spans multiple disciplines or a variety of simpler coupled problems similar to the FEniCS [2] and Firedrake[3] tutorials. **(Not started)**

3.3 August 9, 2019 Milestone

By the start of August, the goal is to have several different classes created within the framework. These classes should capture the complete set of coupled PDEs required for the demonstration problems. These classes will allow users to freely couple problems in the available domains and should be a basic foundation for developers to continue building their

own coupled problems. The outline of the final framework should be evident at this point with the following features:

1. A way to verify the convergence of the solvers would be beneficial. Although it has not yet been discussed, a Manufactured Solution for some simple examples may be appropriate here to test the framework. **(Not started)**
2. To be determined: whether any extra functionality or nice to have features should be included. **(Not started)**

3.4 August 30, 2019 Milestone

This period ends with the final submission deadline. The goal during this period of the project will be to update all of the code documentation and complete the final report.

3.5 September 7, 2019 Milestone

This final milestone period will be used to collect the major accomplishments that will be highlighted in the final presentation.

References

- [1] Mikael Mortensen, Hans Petter Langtangen, Garth N. Wells. A FEniCS-Based Programming Framework for Modelling Turbulent Flow by the Reynolds-Averaged Navier-Stokes Equations *Advances in Water Resources* 2011;34(9):1082-1101. Available from: <https://www.sciencedirect.com/science/article/pii/S030917081100039X?via%3Dihub>. [Accessed 2019]
- [2] Hans Petter Langtangen and Anders Logg. *Solving PDEs in Python*. Springer; 2017. Available from: <https://fenicsproject.org/tutorial/>. [Accessed 2019]
- [3] Firedrake. *Introductory Jupyter notebooks*. Available from: <https://www.firedrakeproject.org/notebooks.html>. [Accessed 2019]
- [4] DHI. *MIKE 21 MIKE 3 Flow Model FM. Oil Spill Module. Short Description*. Agem Alle 5, DK-2970 Horsholm, Denmark. 2017. Available from: https://www.mikepoweredbydhi.com/-/media/shared%20content/mike%20by%20dhi/flyers%20and%20pdf/product-documentation/short%20descriptions/mike213_os_fm_short_description.pdf. [Accessed 2019]
- [5] Walter Frei. *Which Turbulence Model Should I Choose for My CFD Application*. Available from: <https://uk.comsol.com/blogs/which-turbulence-model-should-choose-cfd-application/>. [Accessed 2019]

- [6] Anders Christian Erichsen, Flemming Møhlenberg, Rikke Margrethe Closter, Johannes Sandberg. *Models for transport and fate of carbon, nutrients and radionuclides in the aquatic ecosystem at Öregrundsgrepen*. Svensk Kärnbränslehantering AB: R-10-10, 2010.