Exercise 1

Introduction

This is the first of four exercises to introduce how declarative PDE solvers in python can be easily used to solve PDEs of interest. To get things started, we will work through one of the examples from the problem sheets and present both the theoretical and numerical solution.

Learning objectives

After completing this first exercise, you should be able to do the following:

1. Set up a simple case within `FiPy` (https://www.ctcms.nist.gov/fipy/)
2. Understand what a mesh is in the context of numerical methods
3. Understand the different methods of benchmarking and validating a model and simulation
4. Perform basic visualisation of simulation results using the visualisation tools built into `FiPy`
5. Generate plots for use in a report

Why bother with simulations

The `Math II` course you would be working on now (or possibly have completed) would have introduced a range of important advanced mathematical concepts that are used to study problems of interest to engineers. However, as the emphasis of the course is to convey the mathematical principles in the clearest manner possible, many gory details are often omitted and most of the problems you would have encountered can be solved analytically.

However, moving forward, the complexities arising from real-world problems such as coupled and nonlinear equations or complex geometries make pursuing numerical solutions the only viable way forward (Date, 2005). You might have had previous exposure to solving ODEs in `MATLAB` during other courses and many of the concepts are highly relevant to solving PDEs. `MATLAB` can also be used to solve PDEs and you should certainly explore it if you are interested and possibly more comfortable with the `MATLAB` environment.

I personally advocate for using python environments and open-source solvers for the following reasons:

- Python and the libraries used are open-source which means you will have access to them at anytime even after you graduate as opposed to `MATLAB`
- Being open-source, you have more access to what is under the hood, but documentation may be scarcer and obtaining support more difficult. It is a double edged sword.
- Familiarity and proficiency in python opens up many opportunities to learn and use other libraries such as data analytics and machine learning packages which are immensely useful and powerful

You may end up using commercially available solver packages in the future e.g. `Ansys` or `COMSOL` which are also incredibly powerful and come with a nice graphic user interface. However, the fundamentals are the same and insights gained from one solver are often useful in future practice.

Problem statement

We use the Q3 of problem sheet B2-3 as the model problem here. This equation represents the Laplace equation in $2\mathrm{D}$ along the domain $(x, y) \in [(0, 0), (a, b)]$:

$$\nabla^2 \phi = 0$$

Subject to the following Dirichlet boundary conditions:

$$\phi(0, y) = \phi(a, y) = \phi(x, 0) = 0,$$
$$\phi(x, b) = 1$$

Analytical solution

Analytically, we can solve this problem using the separation of variables. Suppose that the solution to the equation $\phi(x, y)$ takes on the following form:

$$\phi(x, y) = X(x)Y(y)$$

We can recast the PDE as such:

$$\frac{d^2X}{dx^2} + \lambda X = 0$$
$$\frac{d^2Y}{dy^2} - \lambda Y = 0$$

Solving the first ODE:

$$X(x) = c_1 \cos(\lambda x) + c_2 \sin(\lambda x),$$

where c_1 and c_2 are the constants of integration and λ is separation constant which needs to be evaluated. We use the boundary conditions to evaluate the constants and the non-trivial values of λ

Applying the appropriate BC i.e. $X(0) = X(a) = 0$, we end up with:

$$X(x) = \sum_{n=0}^{\infty} c_n \sin(\frac{n\pi x}{a}),$$

where c_n is to be evaluated subsequently. As we are able to write an expression for λ i.e. $\lambda = \frac{n\pi}{a}$, we can consider the 2nd ODE:

$$\frac{d^2Y}{dy^2} - \frac{n\pi}{a}Y = 0$$

Which gives us the general solution:

$$Y(y) = c_4 \cosh\left(\frac{n\pi}{a}y\right) + c_5 \sinh\left(\frac{n\pi}{a}y\right),$$

where c_4 and c_5 are constants of integration. Applying the appropriate boundary condition i.e. $Y(0) = 0$:

$$Y(y) = c_5 \sinh\left(\frac{n\pi}{a}y\right)$$

We therefore end up with the general solution for $\phi(x, y)$:

$$\phi(x, y) = \sum_{n=0}^{\infty} \bar{a}_n \sin\left(\frac{n\pi}{a}x\right) \sinh\left(\frac{n\pi}{a}y\right),$$

where $\bar{a}_n = c_5 \cdot c_n$ and needs to be evaluated by applying the final boundary condition.

We can write $\phi(x, b)$ as follows:

$$\phi(x, b) = \sum_{n=0}^{\infty} \bar{a}_n \sinh\left(\frac{n\pi b}{a}\right) \sin\left(\frac{n\pi}{a}x\right)$$

We are able to evaluate $\bar{a}_n$ as follows:

$$\bar{a}_n \sinh\left(\frac{n\pi b}{a}\right) = \frac{2}{a}\int_0^a \sin\left(\frac{n\pi}{a}x\right)dx$$

$$\bar{a}_n \sinh\left(\frac{n\pi b}{a}\right) = \frac{2}{n\pi}[1 - (-1)^n]$$

For even values of n, $\bar{a}_{2n} = 0$ while for odd values of n:

$$\bar{a}_{2n+1} = \frac{4}{(2n+1)\pi \sinh\left(\frac{(2n+1)\pi b}{a}\right)}$$

We are able to write the solution to our problem:

$$\phi(x, y) = \sum_{n=0}^{\infty} \frac{4}{(2n+1)\pi \sinh\left(\frac{(2n+1)\pi b}{a}\right)} \left[\sin\left(\frac{(2n+1)\pi}{a}x\right) \sinh\left(\frac{(2n+1)\pi}{a}y\right)\right]$$

Numerical solution

We consider the numerical solution of the Laplace equation with the same boundary conditions as above on a unit square domain i.e. $a = b = 1$. The analytical solution can be written as follows:

$$\phi(x, y) = \sum_{n=0}^{\infty} \frac{4}{(2n+1)\pi \sinh((2n+1)\pi)} \left[\sin((2n+1)\pi x) \sinh((2n+1)\pi y)\right]$$

The analytical solution is of interest to us as it gives us an exact solution which we can use to compare with the simulation. This is important as often, simulations can give us unexpected or inaccurate results and we need to be sure if what we are evaluating is correct. Broadly, we can use the following strategies for benchmarking and validating our simulations:

1. Comparison to an analytical solution (What we are doing in this exercise)
2. Comparison to experimental data e.g. Computational Fluid Dynamics (CFD) simulations of real flows which we then go and test experimentally to confirm if we observe the same trends and flow features.
3. Comparison to previous literature and well studied test cases. Many journal articles that propose new techniques for simulations typically use a series of well studied benchmarking

cases that have a lot of data available and compare their results to these test cases.
4. Mesh resolution / convergence study. Instead of comparing the simulation results to other work, a mesh resolution study where progressively finer meshes can be used in the simulation to evaluate if the results are independent of the mesh resolution.

Typically, a combination of these techniques will be used in an actual study.

For the purposes of this course, the more gory details regarding the discretisation schemes and the complexities of solution method are omitted. However, there is a wide variety of resources available that can shed more light if necessary. The `FiPy` website itself contains a wealth of information in a digestible manner and is a great place to start.

Setting up the mesh

We first need to set up the mesh. When we look at the analytical solution of the PDE, we can see that we are able to fully evaluate ϕ at any point (x, y) within the domain $(x, y) \in [(0, 0), (1, 1)]$. However, when solving PDEs numerically, we do not have that luxury as we would then need to evaluate ϕ at infinite points. Instead, the physical domain is represented as discrete chunks, thus representing the domain in a discrete and finite way that can be interpreted by a computer.

`FiPy` has many in-built mesh generation tools for a variety of simple shapes e.g. a line, rectangle or sphere for example. We create the mesh in our code as follows:

```
nx = ny = 100
dx = dy = 0.01
mesh = Grid2D (dx=dx, dy=dy, nx=nx, ny=ny)
```

This gives us a mesh of 100×100 cells with each cell having the dimensions (Δx,Δy) of $(0.01, 0.01)$. `FiPy` uses what is known as finite volumes approach when discretising the mesh (see documentation for more details).

Defining the variable

Next we need to define the variable of interest. ϕ in this case is `CellVariable` which is the most common type you would encounter during such simple problems. A detailed list of the different types of variables available in `FiPy` can be found in :https://www.ctcms.nist.gov/fipy/fipy/generated/fipy.variables.html

The addition of the last portion `value = 0.0` initializes the value of ϕ to 0

```
phi = CellVariable(mesh=mesh, name=r"$\phi$", value= 0.0)
```

Applying the boundary conditions

```
# Applying Dirichlet boundary conditions only
valueTop = 1.0
valueBottom = valueLeft = valueRight = 0.0

# Top BC
phi.constrain(valueTop, mesh.facesTop)
```

```
# Bottom BC
phi.constrain(valueBottom, mesh.facesBottom)

# Left BC
phi.constrain(valueLeft, mesh.facesLeft)

# Right BC
phi.constrain(valueRight, mesh.facesRight)
```

The code itself contains the syntax needed to apply Neumann BCs or combinations of Neumann and Dirichlet BC for you to explore with.

Defining the equation

The way `FiPy` requires you to declare the equation makes it highly intuitive for transport problems. This link contains alot of inf ormation regarding how different terms in an equation can be expressed in `FiPy` and what numerical schemes are available for solution: https://www.ctcms.nist.gov/fipy/documentation/numerical/index.html

In this case, our equation is quite simple:

```
phi.equation = (DiffusionTerm(coeff = 1.0) == 0)
```

Defining the analytical solution

For this first example, we wish to use the default visualizing tools within `FiPy` to see our results. Therefore, the analytical expression needs to be declared in a way that is acceptable:

```
pi = numerix.pi

x = mesh.cellCenters[0]
y = mesh.cellCenters[1]

analytical_solution = ( (4.0/(pi*numerix.sinh(pi)))*(numerix.sin(pi*x))*
(numerix.sinh(pi*y)) +
                    (4.0/(3.0*pi*numerix.sinh(3.0*pi)))*(numerix.sin(3.0*pi*x))*
(numerix.sinh(3.0*pi*y)) +
                    (4.0/(5.0*pi*numerix.sinh(5.0*pi)))*(numerix.sin(5.0*pi*x))*
(numerix.sinh(5.0*pi*y)) +
                    (4.0/(7.0*pi*numerix.sinh(7.0*pi)))*(numerix.sin(7.0*pi*x))*
(numerix.sinh(7.0*pi*y)) +
                    (4.0/(9.0*pi*numerix.sinh(9.0*pi)))*(numerix.sin(9.0*pi*x))*
(numerix.sinh(9.0*pi*y)) +
                    (4.0/(11.0*pi*numerix.sinh(11.0*pi)))*
(numerix.sin(11.0*pi*x))*(numerix.sinh(11.0*pi*y)) +
                    (4.0/(13.0*pi*numerix.sinh(13.0*pi)))*
(numerix.sin(13.0*pi*x))*(numerix.sinh(13.0*pi*y))
)

analytical = CellVariable(mesh=mesh, name=r"$\phi_{Analytical}$", value =
analytical_solution )
```

Considering that the analytical solution is an infinite series, we need to truncate the series. Ironically, this truncation makes the visualized analytical solution less accurate than the actual simulation as you will see later on. There is an alternate way of implementing the analytical solution in `FiPy` which is shown in ex2a.

Solving the equation

For the purposes of this problem, we shall not bother too much with how the solver is set up and we just use the default settings:

```
phi.equation.solve(var=phi)
```

Setting up the viewers

This is the required syntax to set up the default viewer. `FiPy` uses `matplotlib` for this purpose.

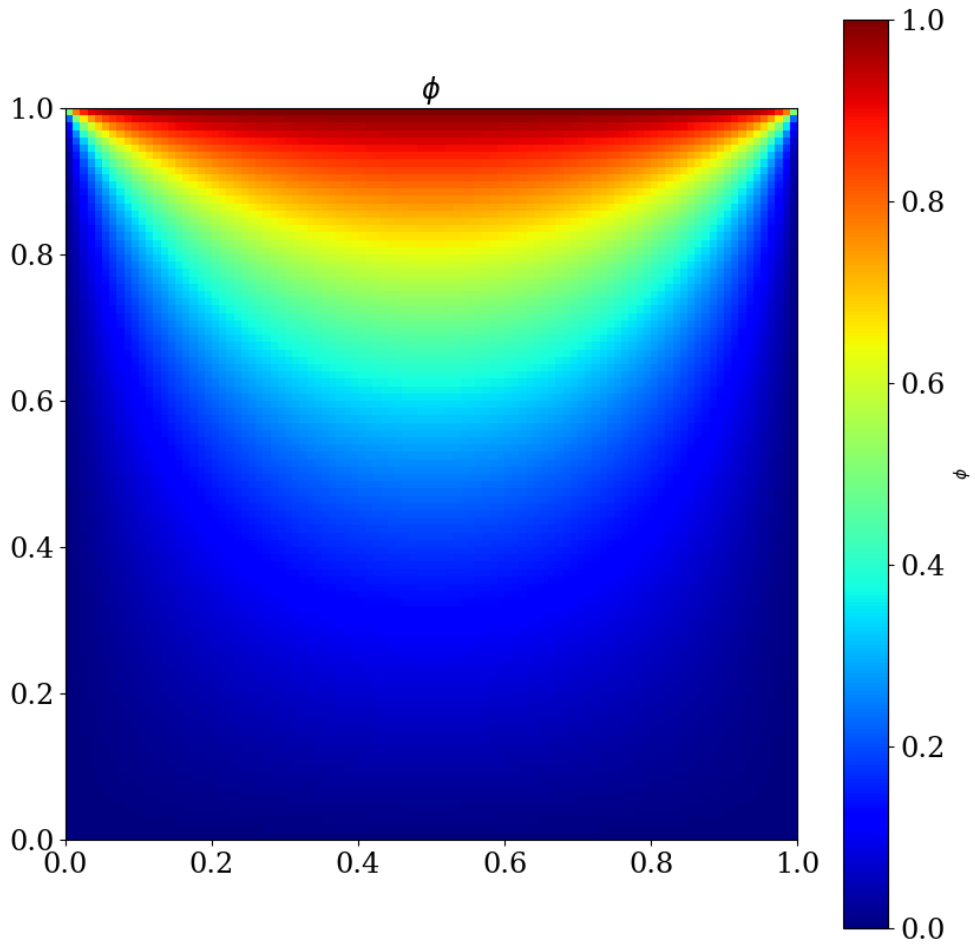
```
from fipy import input
if __name__ == '__main__':
    vi = Viewer(phi, colorbar=None)
    vi.colorbar = _ColorBar(viewer=vi, vmin=0.0, vmax=1.0)
    vi.plot()
    input("Press <return> to proceed")


# Setting up the second viewer for the analytical solution
if __name__ == '__main__':
    vi2 = Viewer(analytical, colorbar=None)
    vi2.colorbar = _ColorBar(viewer=vi2, vmin=0.0, vmax=1.0)
    input("Press <return> to proceed")
```

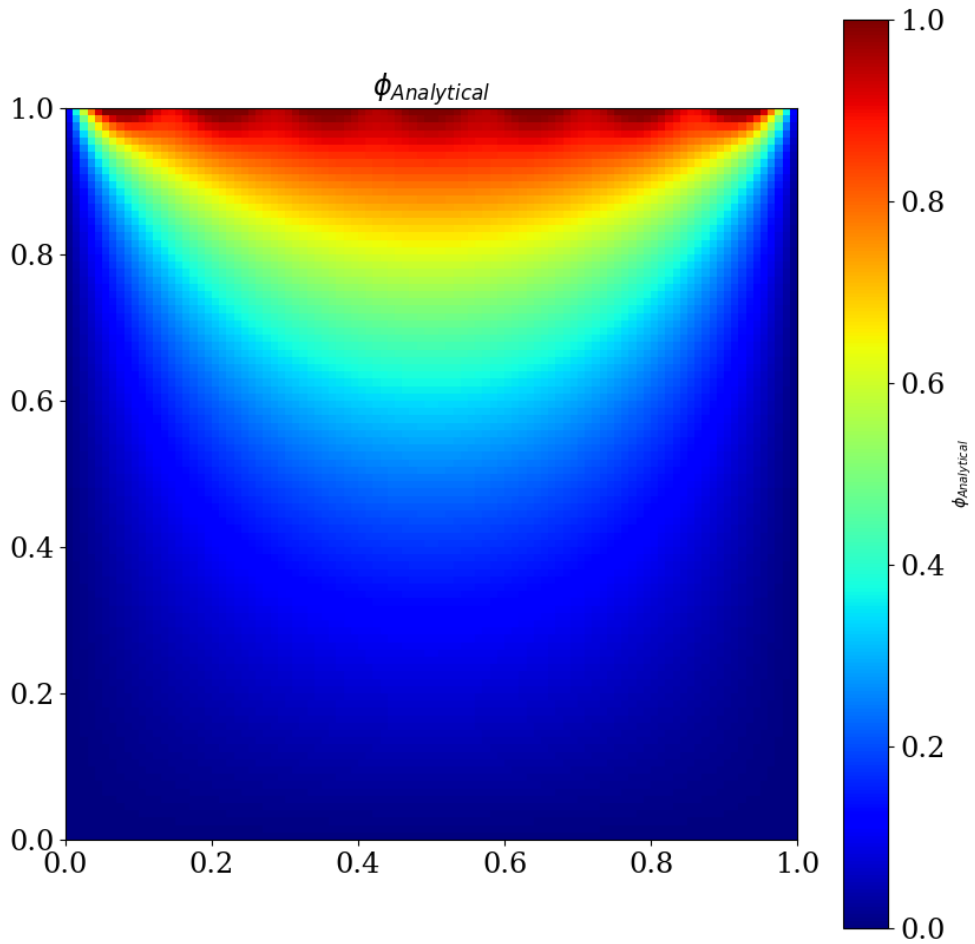
Post processing

Fortunately there is not too much work that needs to be done for this exercise. The data is already nicely scaled and represented using the default viewer settings. The task then is to export the data from the viewer which can be done trivially by clicking the save icon when the viewer comes up during the simulation. We get the following results:

Numerical solution:



Analytical solution:


$\phi_{Analytical}$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
$\phi_{Analytical}$