ENGSCI 331 Computational Techniques 2

Finite Differences Lab - Weeks 1 and 2

Semester Two 2022

Introduction

Background and Lab Objective

This lab will focus on the numerical solution of partial differential equations (PDEs) using the method of finite differences. There will be three assessed tasks to complete.

Lab Files and Code:

The lab files will be made available from `Canvas > Files > FiniteDifferences > Lab`.

Submission Instructions:

Upload your code:

Please combine all code files for your submission into a single zipped directory and upload to the relevant Canvas assignment. Please do **not** modify the original file names or include any additional code files. Please do **not** modify any existing class, method, or function names. You are allowed to create additional classes, methods and functions in the provided functions file.

Your code will be run through software to detect similarities with other student code, so please ensure that all code submitted is your own work.

Hand-In Items:

Some tasks have specific workings that you should hand-in, in addition to your code. These are clearly labelled in this lab document.

Your hand-in items should be compiled into a brief report named `report_upi.pdf/doc/docx`. Please upload this alongside your zipped code (but not as part of the zipped code, as then it is not easily visible on Canvas for marking) as part of your assignment submission.

Any written comments in your report can be targeted at a reader who has a complete understanding of the methods and techniques, and knows what you have been asked to do. It should detail anything interesting, unexpected, or complex in the implementation; present and very briefly discuss interesting results; and present any suitable conclusions if appropriate.

Task 0: 2D Poisson Equation with Dirichlet BCs

Background and Objective

Note: This task will **not** be directly assessed. However, the class that we are constructing in Task 0 will be re-used in Task 1 (the first assessed task), which will be released in the second week of the module once more has been covered. It is therefore recommended that you focus on completing Task 0 in the first week of this module.

A rectangular well has achieved an equilibrium pressure distribution, $u(x, y)$, that can be modelled mathematically by the following partial differential equation (PDE):

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 6xy(1-y) - 2x^3$$

for $0 \leq x \leq 1$ and $0 \leq y \leq 1$. It is subject to the Dirichlet boundary conditions:

$$u(0, y) = 0 \ , \quad u(1, y) = y(1-y) \ , \quad u(x, 0) = 0 \ , \quad u(x, 1) = 0$$

It is possible to solve this PDE analytically to give:

$$u(x, y) = y(1-y)\,x^3$$

The objective of this task is to construct a finite difference solver for the Laplace/Poisson equation in 2D Cartesian coordinates with only Dirichlet boundary conditions. The solver should have **all** mesh points, including those along Dirichlet boundary conditions, as part of the system of linear equations $A\vec{u} = \vec{b}$, to be solved.

It is important that $\vec{u}$ be structured in a logical way, for example in the way shown in lectures (start at (x_0, y_0), with inner iteration across x and outer iteration across y). If you choose a non-standard way of ordering mesh points in u, it will be difficult to reshape between 1D and 2D, and also more difficult for you/us to test that it has been constructed correctly.

We will be using an object-oriented programming (OOP) approach, constructing a class that can be applied easily to different 2D Cartesian Poisson/Laplace problems. This approach has the added advantage of retaining key information alongside the solution, such as the mesh spacing used.

Steps to Complete

1. Complete the method `def __init__` in `class SolverPoissonXY`.

   This method will initialise useful attributes for the class, such as:

   - The total number of mesh points, and number along each dimension.

- The uniform mesh spacing.
- The mesh x and y coordinates along each dimension.
- The boundary conditions.
- The Poisson function (equal to zero for Laplace equation problems).
- The matrices required for solving $A\vec{u} = \vec{b}$.

2. Complete the method `def dirichlet` in `class SolverPoissonXY`.

   This method will update the corresponding elements of A and $\vec{b}$ for the Dirichlet boundary mesh points **only**. For Task 0, all rectangular boundaries have Dirichlet conditions. In Task 1, we will consider a problem with two different types of boundary conditions: Dirichlet and Neumann. Therefore, this method should also check that the dictionary for a boundary specifies that it is of type `'dirichlet'`.

3. Complete the method `def internal` in `class SolverPoissonXY`.

   This method will update the corresponding elements of A and $\vec{b}$ for the internal mesh points. This will include applying the five-point finite difference stencil (shown in lecture) to A, and the Poisson equation to $\vec{b}$. We can assume that, if applying this solver to the Laplace equation, that the Poisson function is simply equal to zero for any input x and y i.e. we can use this same class for both the 2D Poisson and Laplace equations.

4. Complete the method `def solve` in `class SolverPoissonXY`.

   This method will call `def dirichlet` and `def internal` to form the system of linear equations, $A\vec{u} = \vec{b}$. The built-in NumPy linear algebra solver `np.linalg.solve` can then be used to solve for $\vec{u}$. The built-in NumPy matrix re-shaping function `np.reshape` can be used to apply the 1D $\vec{u}$ to the 2D mesh solution, `self.solution`.

5. **Optional:** Write a test case that checks your solver is working as expected. We have the analytical solution available, so we can determine the accuracy of our numerical solution.

6. Complete the method `def plot_solution` in `class SolverPoissonXY`.

   This method will plot the mesh solution as a 2D contour plot. The built-in NumPy function `np.meshgrid` may be useful for setting the x and y coordinates of each mesh point. Include a colour bar for the contour plot. Check carefully that the plot has the correct orientation relative to the boundary conditions. If necessary, rotate with NumPy built-in functions.

Hand-In

There are no specific hand-in items for this task.

Task 1: 2D Poisson Equation with Mixed BCs

Background and Objective

A rectangular well has achieved an equilibrium pressure distribution, $u\left(x,y\right)$, that can be modelled mathematically by the 2D Poisson equation:

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=x-y$$

subject to the following Neumann and Dirichlet boundary conditions:

$$\begin{aligned}\frac{\partial u}{\partial x}\left(-2,y\right)&=x\\ \frac{\partial u}{\partial x}\left(2,y\right)&=y\\ u\left(x,-3\right)&=xy\\ u\left(x,3\right)&=xy-1\end{aligned}$$

The objective of this task is to further develop the finite difference solver from Task 0 to also solve the Laplace/Poisson 2D Cartesian coordinates with mixed Dirichlet and Neumann boundary conditions.

Steps to Complete

If you have not already completed Task 0, the "steps to complete" from that task can be completed prior to completing the below steps for Task 1. A clarification with regards to the attribute `solution` of `class SolverPoissonXY`, make sure to read the provided docstring carefully as to what it represents (2D solution on mesh or 1D solution in $A\vec{u}=\vec{b}$).

1. Complete the method `def neumann` in `class SolverPoissonXY`.

   This method will update the corresponding elements of A and $\vec{b}$ for the Neumann boundary mesh points.

   The solver class should be able to account for a single Neumann boundary condition on either the top, bottom, left or right, or a pair of Neumman boundaries on opposing sides (as in this problem). It can be assumed that the class does not need to consider a situation where a corner mesh point is subject to two Neumann boundary conditions i.e. all corner mesh points should be treated as subject to a Dirichlet boundary condition.

2. Update the method `def solve` in `class SolverPoissonXY`.

   This method should now also call your completed `def neumann` as part of forming the system of linear equations, $A\vec{u}=\vec{b}$.

Hand-In

1. Contour plots of the solution for two different options for the mesh spacing:

   - $\Delta_a = 2$
   - $\Delta_b = \frac{1}{10}$

   Write a brief comment on the relative accuracy and computational expense of these two different solutions.

Task 2: 1D Heat Equation

A student is designing a high powered heating rod, which is 5 metres in length, for heating up a tank of liquid via convection. Before the rod is heated, it has a uniform temperature of 20 C (approximately room temperature). The two end points of the rod are then increased near-instantaneously to a temperature of 200 C. The student is interested to know which, if any, of the rods will have a temperature that exceeds 172 C across the full length of the rod within 4 seconds of the ends being heated.

The flow of heat within the rod can be modelled using the 1D heat equation:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

where x is the distance along the rod in metres, t is the time elapsed in seconds and α is a measure of the thermal diffusivity of the rod material. The student has three rods to choose from, each with a different thermal diffusivity:

Material	α
Silver	1.5
Copper	1.25
Aluminium	1

Dirichlet boundary conditions can be used for the rod ends:

$$u(0,t) = 200$$
$$u(5,t) = 200$$

After the near-instantaneous heating of the rod ends to 200 C, the initial conditions for the rod temperature can be represented using the piecewise function:

$$u(x,0) = \begin{cases} 200 & x = 0 \\ 30 & 0 < x < 5 \\ 200 & x = 5 \end{cases}$$

The objective of this task is to implement both an explicit and implicit finite difference solution method for this PDE, within a single solver class, and to use these to choose a suitable material to use for the rod.

Steps to Complete

1. Copy the template code for `class SolverHeatXT` from file "class_solver_heat.py" into your "functions_fd.py" file and work on it there. I want students to submit only a single functions file containing code for all classes we write in this lab.

2. Complete the method `def __init__` in `class SolverHeatXT`.

   This method will initialise useful attributes for the class. Look for the "TODO" statements within to get more specific details on what to complete.

3. Complete the method `def solve_explicit` in `class SolverHeatXT`.

   This method will implement an explicit solution method for the 1D heat equation.

4. Complete the method `def implicit_update_a` in `class SolverHeatXT`.

   This method will calculate the coefficients in A, based on the weighting θ (stored in attribute `self.theta`) applied to the spatial derivative at t_{n+1} i.e. $0 < \theta \leq 1$. Note that A only needs to be set once and does not need to be updated each time step.

5. Complete the method `def implicit_update_b` in `class SolverHeatXT`.

   This method will update $\vec{b}$ as part of solving the current time step, using the data already stored in `self.solution`. This method will need to be called for each new time step.

6. Complete the method `def solve_implicit` in `class SolverHeatXT`.

   This method will call `def implicit_update_a` to set the coefficients in A based on the value of θ. It will then iteratively call `def implicit_update_b` to update $\vec{b}$ and solve the current time step. A built-in linear algebra solver, such as `np.linalg.solve`, may be used as part of this iterative process.

7. Complete the method `def plot_solution` in `class SolverHeatXT` class.

   This method will plot the solution at a few regularly spaced values of time. It is up to you to select an appropriate number of time steps at which to plot the solution.

Hand-In

- For a spatial mesh spacing of $\Delta x = 0.1$, determine an expression for the largest possible temporal mesh spacing, Δt, that will guarantee numerical stability of an explicit solution method for all three rods.

- For the silver rod, solve the following computational models over a time interval $0 \leq t \leq 4$:

  - Explicit method with $\Delta x = 0.1$ and $\Delta t = 0.001$.

  - Explicit method with $\Delta x = 0.1$ and $\Delta t = 0.005$.

  - Crank-Nicolson implicit method with $\Delta x = 0.1$ and $\Delta t = 0.001$.

  - Crank-Nicolson implicit method with $\Delta x = 0.1$ and $\Delta t = 0.005$.

Provide a suitably labelled plot for each solution. Comment briefly o n w hich o f these numerical models are suitable for use in analysing the thermal properties of the rod.

- Use an appropriate computational model to solve the system for each of the three rods over a time interval $0 \leq t \leq 4$. Identify which, if any, of the rods exceed a temperature of 172 C across the entire rod after four seconds have elapsed from the ends being heated to 200 C.