ENGSCI 331 Computational Techniques 2

Finite Differences Lab - Week 1

Semester Two 2023

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`:

- `task[...]_fd.py` - a script file for each Task.
- `functions_fd.py` - contains the classes and functions used throughout the tasks.

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 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.