BEng Biomedical Engineering
Computational Methods

Coursework Partial Differential Equations - Take-Home Assignment

Objective

To gain practical experience in writing code that computes a numerical solution of a PDE.

Introduction

In two dimensions, Laplace's Equation can be written

$$u_{xx} + u_{yy} = 0, \text{ for } (x, y) \in \Omega$$

where Ω is the Region of Interest (ROI) inside which we want to solve the equation. In short form it can be written $\Delta u = 0$ or $\nabla^2 u = 0$. Laplace's Equation needs a boundary condition to be *well-posed*. A Dirichlet boundary condition is sufficient, i.e. one that specifies the value of u at all points on $\partial\Omega$. In other the words, the value of $u(x, y)$ needs to be given for all points (x, y) that are on the boundary of the ROI $\partial\Omega$.

For this coursework task, you need to write code that uses a numerical scheme to estimate the solution for Laplace's Equation for a ROI in a discrete grid of points. You will be given files that contain the following information

- Which grid points are on the boundary of the ROI and which are inside.
- The values of the function at the boundary points.

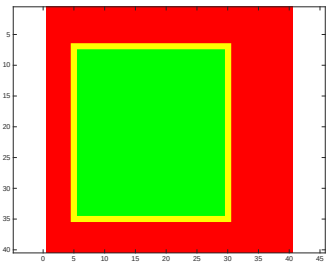
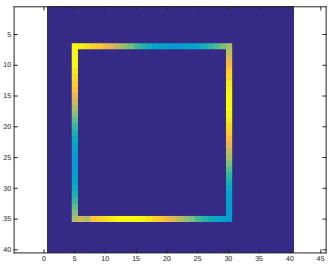
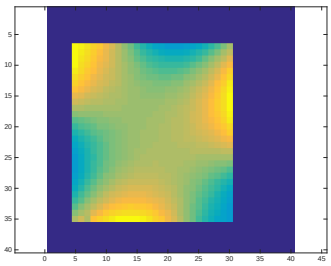
In other words, the files describe the ROI and the boundary condition. Your code should read these files and then use iterative Jacobi methods to estimate the solution of Laplace's Equation inside the ROI.

Instructions

You will be given two examples of problems to solve (i.e. the ROI information and boundary values for each). You should check that your code works with each one. When you are satisfied that your code is working, you will make up your own ROI and boundary values and test that your code still works.

The data files for the example problems can be downloaded from KEATS. These include `ROI_1.txt` and `bdry_Values_1.txt` for the first example problem. Each file contains an array of data. The left hand image below illustrates the contents of the file `ROI_1.txt` which describes a rectangular ROI. This is in the form of a 40×40 array with values of $+1$ for points inside the ROI (green in the diagram), 0 for points on the boundary (yellow), and -1 for points outside the boundary (red)[1].

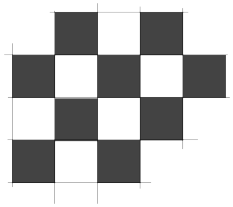
[1]The arrays are displayed using the `imagesc`.

You can assume that the points of the array (including those outside the ROI) correspond to a regular grid of points in the unit square $\{(x, y) : 0 \leq x, y \leq 1\}$. The middle image above shows the contents of the file `bdry_Values_1.txt`, another 40×40 array where the boundary condition values are specified. The values in this array that do not correspond to boundary points can be ignored.

The right hand image shows the result of solving Laplace's equation inside the rectangular ROI with the given the boundary conditions. You can use this to check if your solver is working. A second pair of files, `ROI_2.txt` and `bdry_Values_2.txt` are given for a second example problem where the ROI is in the shape of a circle and the number of points in the grid is different from problem 1. You should also use this problem to test your code.

In the notes and slides, you already have pseudocode to help you write MatLab code for solving a PDE with Jacobi iteration. In this pseudocode, the ROI points are updated during each iteration in a separate copy of the solution array. This method will be called **Method A**.



An alternative method updates the ROI points of the solution array *in place*, i.e. not in a separate copy. This can done in two *passes* for each iteration. If we consider the ROI points as a chess board, then, during each iteration, the white squares are updated on the first pass and the black squares on the second pass. This will be called **Method B**.

Your completed program should:

- Read in data files for a given problem's ROI and boundary values, storing this information in a suitable variable(s).
- Set the convergence criteria to be used (i.e. when to stop iterating).
- Solve Laplace's equation for the problem using both methods A and method B.
- Visualise the solutions obtained by each method[2].
- Give an output message(s) for both methods' performance in terms of the number of iterations and the time taken.

There are likely to be differences between the solutions provided by methods A and B but these should be small.

For this assignment you should also make up a third problem of your own (Problem 3). Generate its data files and call them `ROI_3.txt` and `bdry_Values_3.txt`. Again, test that both your solvers work on this problem. Note how the boundaries given in the examples provided are *closed* and the the values on the boundary vary in a reasonably *continuous* way (i.e. with no large changes between nearby boundary points).

[2]You can use any method for this, `imagesc` was used above but you can also use `surf`, `contour`, or anything else that is suitable.

Before starting any coding, you should produce a design for your program, using a structure chart(s). You should then apply an incremental development approach to develop your code. There are many ways that this exercise can be tackled. A good approach is to break the tasks that you need to do down so that they can be implemented in a number of smallx functions that are called by a single main function or script.

Reporting Requirements

You should submit

- MatLab **script/function (*.m*) files** that meet as many of the requirements as possible.
- The data files for your own problem 3: `ROI_3.txt` and `bdry_Values_3.txt`
- You should also submit a **structure chart(s)** for your program design and include a very brief description of the parts of your design and how they relate to each other. To help you with this, have a look at the structure charts from the Introduction to Computer Programming Module (the examples on KEATS or in the notes).

In your code, remember to use good names for variables and functions and make good use of comments[3] and indentation to make your program as easy to understand as possible. It is important that your code is clear and easy to read.

Submission will be via the KEATS system. The submission point will only allow you to upload a single file so you should combine all files into a single *zip* file and submit this *zip* file.

Name your file *cw-PDEs-SURNAME-FIRSTNAME.zip*, replacing *SURNAME* and *FIRSTNAME* with your personal details (e.g. *cw-PDEs-ALJABAR-PAUL.zip*). The hand-in date is **27 Feb 2023, 5 pm**. Late submissions (within 24 hours of this deadline) will be accepted but will be capped at the module pass mark (i.e. 40%).

If your program does not meet all requirements then please submit what you have written by the deadline.

This is an assessed assignment. The work that you submit **must be your own work** and not anyone else's work. Even if your work is incomplete, then please submit what you have done by the deadline to get credit for the parts you have finished.

Assessment

Your coursework will be marked on a number of factors:

- Does the code work? Does it meet all computational methods requirements? (65%)
- Is the code well designed, are the different components of the method well divided into appropriate functions? (20%)
- Is the code clear and easy to understand (15%)

This is an individual assignment. You are not permitted to work together with any other student.

If you have any questions about this coursework please contact Pablo Lamata (pablo.lamata@kcl.ac.uk).

[3] N.B. Good use of comments does not necessarily mean *lots* of comments.