ENG3165 – Coursework – 2015/16

Part 1 of 2 :: Numerical Methods

This is only part 1 of 2 of the ENG3165 coursework. Make sure to check the instructions for part 2 (CFD) when available

Instructions – IMPORTANT: please read carefully

1. Submit your coursework paper (e.g a Word, OpenOffice or PDF document), along with all the other requested files in your "NM & CFD Coursework" assignment folder on **SurreyLearn**. The deadline is **Tuesday 15th December at 16:00**.
2. Any evidence of **plagiarism** or **collusion** with others will be dealt with severely under the University rules. Please ensure that what you submit is your work and only your work; do not 'assist' or seek assistance from others by e.g. 'sharing' parts of programmes, and do not under any circumstances copy code from books/websites. (It is actually very easy to tell the work of relatively novice programmers such as yourselves from that of more sophisticated and experienced programmers!).
3. Several questions will require using data values that are different depending on you URN, and in particular, two of its last three digits. Make sure you use the right values from your **URN digit**. Double check them before submitting. If you fail to use the correct URN values, we are afraid we cannot help.
4. For this part 1 (Numerical Methods) your coursework paper must include the **Matlab code** itself (e.g. copy/pasted from the Matlab Editor); in addition, your Matlab code must be **also** submitted on SurreyLearn as **separate m-file**s.
5. The coursework is a mix of easy MATLAB programming and some more advanced programming. The awarded marks are clearly specified at the end of each question.
6. On each of the elements of the questions, marks will be awarded for:
    a. A basic **functioning** program that **satisfies the minimum requirements** and produces **correct answers**, with output of the key final results: such a program will earn from the pass mark on this element to about 75-80% depending on what is included.
    b. Good program layout and well-commented (i.e. sufficient comments to guide the user but not excessive and cluttering) programs so that the "reader" can identify variables, and understand the processes: this will be worth between 10 and 25% of each program element
    c. When required (Questions 4, 6 and 8) a good, but **brief**, description or explanation of the reasoning behind your code (NOTE: this is in addition to the commenting within the code itself). This will be worth up to 40%, depending on the question. This item will include the derivation of the discretised equations in Questions 6 and 8.
    d. For the more complex programs (Questions 7 and 8), good presentation of results and more sophisticated programming to improve the 'basic' program level will earn more marks.

7. The marking scheme for Part 1 (NM) is shown in the following table:

Marks \ Questions	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Basic Matlab	1.5	1.5	1.5	2	4	2.5	2	2.5
Layout and commenting	0.5	0.5	0.5	0.5	1	0.5	0.5	0.5
Description / derivation	0	0	0	1.5	0	2	0	2
Presentation	0	0	0	0	0	0	1.5	1

8. Most of the questions **can be answered independently**, without the need to successfully complete the previous ones. The only exception is the question that asks you to make use of all the previous pieces of code, i.e. question 7.
9. As part of this coursework you will be required to write and submit several MATLAB programs. As the University has a site licence for MATLAB, if you do not already have a copy, or have access to a copy, you should be able to get one for personal University use. To obtain this go to http://info.eps.surrey.ac.uk/IT/software/matlab/ and follow instructions. Please contact IT services if you have a problem: I am afraid I cannot help with this.
10. Check that your MATLAB files are free of errors and they do exactly what they are supposed to do is **very important**. Chances are you will spend more time checking for errors and double-, triple- check that everything is all right than actually coding from scratch. This is not an uncommon situation in "real-life" programming jobs… Functions need to have **exactly** the variables and arguments specified in the questions, **exactly** in the requested order.
11. Use the MATLAB file names specified **exactly** as requested, including capital letters, symbols and numbers – if present. Any deviation, even very small, will sensibly slow down the marking process

Data for use in coursework – part 1:

Some of the questions in this coursework require use of various digits from your URN. Please complete the grid below and then when the question references digits ***a*** or ***b*** then these should be obtained from the corresponding digit in the grid.

The letters below the boxes will be used in the coursework: where they appear you should substitute the digit in the box immediately above.

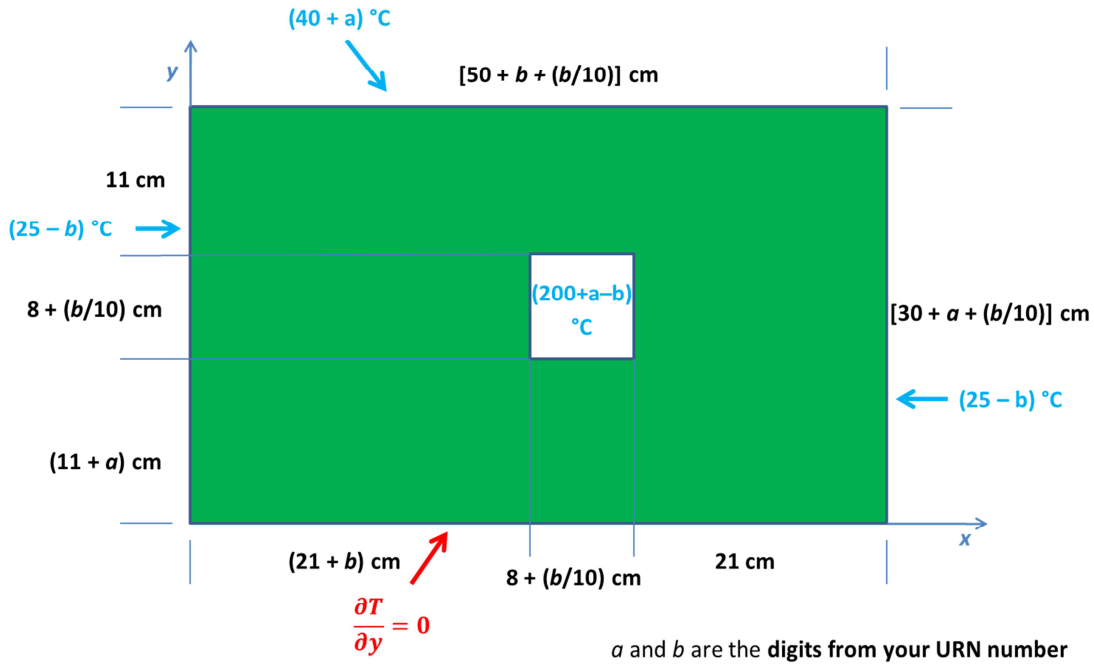
URN:

x	**x**	**x**	**x**	**a**	**b**	**x**

Part 1 (Numerical Methods) [30 marks]

Case study:

Let's consider a rectangular plate with a square hole in it:



Assuming the plate is very thin (thickness is negligible) and it is insulated both above and below the two flat faces, we want to calculate the distribution of temperature within the plate.

A metallic bar goes through the square hole, maintaining the temperature there at (200 + *a* – *b*) °C, while at the plate edges the temperature is fixed at (25 – *b*) °C (right and left sides) and (40 + *a*) °C (top side), where *a* and *b* are the **digits from your URN number**. The bottom side of the plate is insulated so that the boundary condition for that particular edge is $\frac{\partial T}{\partial y} = 0$.

Initially we'll consider the steady-state case. As usual, the steady-state temperature distribution within the plate can be modelled mathematically through the Laplace equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \qquad (1)$$

There are several ways of discretising the problem, some more accurate than others. However, the easiest way is to use a uniform grid over the whole plate, **including** the square hole. The temperatures at the computational nodes that fall within the square hole can then be treated as

additional boundary conditions: their temperature will be fixed at (200 + *a* – *b*) °C, instead of using the discretised equation for the normal internal nodes.

In this way, apart from the boundary nodes, we can discretise using the Finite Difference Method and, for example, the Central Difference Scheme obtaining, for the internal nodes:

$$\frac{1}{\Delta x^2}\left(T_{i+1,j} + T_{i-1,j}\right) - 2\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)T_{i,j} + \frac{1}{\Delta y^2}\left(T_{i,j+1} + T_{i,j-1}\right) = 0 \qquad (2)$$

The following questions will ask you to write Matlab code. Your answers must include the Matlab code itself and, when requested, a description of how the code will work and the derivation of the underlying equations (when applicable). Include them both in your coursework paper and, in addition to it, please also submit the code as m-files.

1) Write a simple MATLAB script to set up the geometry and the computational grid. In particular, you should set up the plate overall dimensions (x and y widths), the x and y coordinates of the lower-left corner of the square hole and the dimension of the square hole (since it is a square, a single value will be enough). These geometrical values can be simply assigned, using the values given in the case study description (see figure). Then ask the user to input the number of nodes along x (ni) and the number of nodes along y (nj) from the keyboard. Save the script with the file name: "**case_setup.m"**.

[2 marks]

2) Write a function to calculate the step sizes along x (Δx), along y (Δy) and the total number of nodes, given the overall dimensions and the number of nodes in each coordinate. In particular the function header should be:

```
function [dx, dy, n] = grid_points(x_width, y_width, ni, nj)
```

where `ni` is the number of points along x, `nj` is the number of points along y, `n` is the total number of points, `x_width` is the width of the plate along x, `y_width` is the width of the plate along y, `dx` is the step size along x and `dy` is the step size along y.

[2 marks]

3) Write a script to ask the user for the boundary values. The script should ask for three values (one temperature at the left and right external edges of the plate, one at the top external edge, and one temperature at the internal square hole) and store them in an array. For example: `bc(1)` will contain the temperature value at the left/right edges, while `bc(2)` will contain the temperature value at the top edge and `bc(3)` the temperature at the hole. Save the script with the file name: "**boundaries.m**"

[2 marks]

4) In questions 1, 2 and 3 we have set up the geometry, the computational grid and the boundary conditions of our case study. Now we want to build the computational matrix (that is, the system of linear equations resulting from the finite difference approximation we used). In order to do this, we start from building a matrix A and a constant vector b. For the time being we do not want to worry about the hole in the plate nor about the bottom insulated side: considering all the nodes that are not at the edges as internal nodes – in this way we can just consider equation (2) for all the internal nodes and deal with both the nodes within the hole and the ones at the bottom at a later stage. The code for doing this is actually very similar to the Thin Plate example discussed in lectures and tutorials. Unfortunately our first attempt is not very successful! We produced the following MATLAB code:

```
function [A,b] = initial_system(ni, nj, dx, dy, T_leftright, T_top)
% create the computational matrix A x = b using FDM (CDS)
%

A = zeros(n, n);
b = zeros(n, 1);

index = reshape(1:n, nj, ni)'; % to convert from row and col to node

for i = 1:ni
    for j = 1:nj
        C = index(i, j); % the index of the node at row i and col j
        if i == 1 || i == ni
            % this is a left/right boundary node
            A(i, i) = 1;
            b(i) = T_leftright;
        elseif j == 1 || j == nj
            % this is a top/bottom boundary node
            A(i, i) = 1;
            b(i) = T_top;
        else % this is an interior node
            % indices for North, East, West and South
            N = index(i, j+1); S = index(i, j-1);
            E = index(i+1, j); W = index(i-1, j);
            A(i, i) = -2*(1 / dx^2 + 1 / dy^2);
            A(i, [N, S]) = 1 / dy^2;
            A(i, [E, W]) = 1 / dx^2;
        end
    end
end
```

The above code is not working. To make it work we must correct two errors (remember: do not take into account the fact that the nodes at the bottom of the plate are not treated correctly; this will be corrected later – now focus on other things). The first one is rather obvious, and can be corrected by adding a single line of code. The second error is a bit more subtle, and must be corrected by modifying the function in several places.

Correct the above function and save it as "**initial_system.m**". However, you must not alter the header (meaning: the return variables, function name and arguments must remain as specified). Additionally, describe briefly what you've changed from the above code and why.

[4 marks]

5) Now that we have set up the initial system of linear equations we want to modify it, to take into account the fact that there is a hole in the plate. What we need to do is to check whether a node is within the hole and, if that is the case, modify the corresponding row in the A matrix and the b vector. In particular, we want to write the following function:

```
function [M, c] = apply_hole(A,b,x_width,y_width,ni,nj,x_hole,y_hole,dim_h,T_hole)
```

where A is the initial matrix, b the initial constant vector, `x_width` and `y_width` are the plate dimensions, `ni` and `nj`, respectively, the number of grid points in the x and y direction, `x_hole` and `y_hole` are the coordinates of the lower-left corner of the hole, `dim_h` is the length of the sides of the square hole and T is the temperature value at the hole. `M` and `c` are the modified matrix and constant vector to be returned by the function. The resulting M and c variables will be copies of A and b, respectively, except for the fact that the equation for the nodes inside the hole must be:

$$T_{i,j} = T_{hole} \tag{3}$$

[5 marks]

6) In a similar way as in question 5, we now want to modify the system of linear equations to take into account the fact that the bottom boundary is insulated. What we need to do is to check whether a node is within the bottom boundary and, if that is the case, modify the correspondent row in the A matrix and the b vector. Before doing this we must also figure out and derive the right equation to use for those nodes.

In particular, we want to write the following function:

```
function [M, c] = apply_insulation (A, b, ni, nj, dx, dy)
```

where A is the current computational matrix, b the current constant vector, `ni` and `nj`, respectively, the number of grid points in the x and y direction, `dx` and `dy`, respectively, the step sizes in the x and y direction. `M` and `c` are the modified matrix and constant vector to be returned by the function. The resulting M and c variables will be copies of A and b, respectively, except for the fact that the equation for the nodes on the bottom boundary must be the one you've derived. Besides the code, please describe how you've derived the equations for the nodes on the bottom boundary.

[5 marks]

7) Now we have most of the elements in place to build a complete script to solve the initial case study. Write such a script using the pieces of code (copy/paste) produced in questions 1 and 3 and using the functions produced in questions 2, 4, 5 and 6 (To be more specific: do not copy/paste the code from the functions; you have to "call" the function from the script). Add a final section to solve the resulting system of linear equations and present the results in a nice way (e.g., by using a contour plot). Save the script as: "**plate_hole.m**".

[4 marks]

8) If you were successful with the previous question, you now have a Matlab script to calculate the steady-state temperatures in the plate. In question 8 we want to modify (or write from scratch) the

code so that we could consider the time-evolving case, starting from an initial state and calculating how the system evolves.

The conservation equation for the 2-D time-evolving system is:

$$\frac{\partial T}{\partial t} = \alpha\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \tag{4}$$

Where α is the coefficient of thermal diffusivity (let's set α = 0.835 cm^2/s). So, we now have three independent variables, instead of 2. Try to discretise the above PDE and write the appropriate code to solve this more complex case study.

Remember that, since we are using an explicit method, we have a **stringent stability criterion**. In this particular case the criterion is:

$$\Delta t \leq \frac{1}{8}\frac{(\Delta x)^2 + (\Delta y)^2}{\alpha} \tag{5}$$

The time step Δt should then be set accordingly. Use T = 0 °C as the initial value of the temperature in the plate (except the hole that will always be, of course at (200 + *a* – *b*) °C.

To present the results, plot a contour plot at selected time steps (e.g. one at 5 s, one at 20 s, one at 100 s and one at 500 s). Save the script as: "**plate_hole_evolving.m**". Besides the code, please describe how you've derived the discretised equations.

[6 marks]