MAE3456 – MEC3456 LAB 06

Due: 11:00PM (Sharp), ~~Monday 8th~~ Wednesday 10th June 2020 (Mon Week 12)

This lab should be completed INDIVIDUALLY. Plagiarism will result in a mark of zero. Plagiarism includes letting others copy your work and using code you did not write yourself. Collaborating with others to discuss algorithms and details of MATLAB syntax and structures is acceptable (indeed encouraged), however you *MUST* write your own MATLAB code. All assignments will be checked using plagiarism-detecting software and similarities in submitted code will result in a human making a decision on whether the similarity constitutes plagiarism.

INSTRUCTIONS

Download **`template.zip`** from Moodle and update the M-Files named **`Lab06_Q1a.m`**, **`Lab06_Q1b.m`**, etc… with your Lab code. DO NOT rename the M-Files in the template and ONLY modify **`run_all.m to add in running scripts that answer any bonus questions I might ask`**. Once you have coded, check your solutions to the questions by running **`run_all.m`** and ensuring all questions are answered as required.

SUBMITTING YOUR ASSIGNMENT

Submit your assignment online using Moodle. You must include the following attachments:

A **ZIP** file (**NOT .rar or any other format**) named in the following way:

Surname_StudentID_Lab_06.zip (e.g. Rudman_23456789_Lab_06.zip)

The zip file should contain the following:

a. All MATLAB m-files for lab tasks: **run_all.m**, **LabNN.m**, **`Q1a.m`**, **`Q1b.m`**, etc…

b. Any additional MATLAB function files required by your code

c. All data files needed to run the code, **including any input data provided to you**

d. Any hand calculations or written responses asked for - scanned in as a ***SINGLE*** PDF file

YOUR ZIP FILE WILL BE DOWNLOADED FROM MOODLE AND ONLY THOSE FILES INCLUDED IN YOUR SUBMISSION WILL BE MARKED

We will extract (unzip) your ZIP file and mark the lab based on the output of **run_all.m** and any hand calculations or written responses. It is your responsibility to ensure that everything needed to run your solution is included in your ZIP file.

STRONG RECOMMENDATION: After uploading to Moodle, download your Lab to a NEW folder, extract it and ensure it runs as expected. ***CHECK YOUR PDF IS INCLUDED***. You can upload your submission as many times as you like ***BEFORE*** the submission deadline. After this time, it is not possible to resubmit your lab without incurring a late penalty. If you forget to include some of your code or your PDF you cannot include it later without penalty.

MARKING SCHEME

This lab is marked out of 95 and full marks is worth 8% of your total Unit mark for the semester. Code will be graded using the following criteria:

1) **`run_all.m`** produces results **automatically** (no additional user interaction needed except where asked explicitly – NOTE, I have included pause commands in run_all so that intermediate answers can easily be viewed by the demonstrators – please don't remove them)
2) Your code produces correct results (printed values, plots, etc…) and is well written.
3) Programming style, efficiency of algorithm and quality of output (figures, tables, written text ...)

ASSIGNMENT HELP

1) You can ask questions in the Discussion Forum on Moodle
2) Hints and additional instructions are provided as comments in the assignment template M-Files
3) Hints may also be provided during lectures
4) The questions have been split into sub-questions. It is important to understand how each sub-question contributes to the whole, but each sub-question is effectively a stand-alone task that does part of the problem. Each can be tackled individually.
5) I recommend you break down each sub-question into smaller parts too, and figure out what needs to be done step-by-step. Then you can begin to put things together again to complete the whole.
6) To make it clear what must be provided as part of the solution, I have used bold italics and a statement that (usually) starts with a verb (e.g. ***Write a function ..., Print the value...***, etc.)

QUESTION 1 [20 MARKS TOTAL]

Background

The 1D linear advection equation is given by:

$$\frac{\partial q}{\partial t} + u\frac{\partial q}{\partial x} = 0 \qquad \text{Eqn(1)}$$

where q is the advected quantity such as heat, and u is the velocity.

You have learned in your lecture that the FTCS scheme for solving the advection equation is unconditionally unstable due to the presence of a numerical diffusion term within the FTCS scheme. One way to overcome this is by using the implicit BTCS scheme or the **backward in time, centred in space (BTCS)** scheme.

Q1a

Write the **BTCS scheme** for the advection equation in Eq 1.

Q1b

Undertake a von Neumann stability analysis for the BTCS scheme in Q1a and determine its stability criteria. Consider just ONE Fourier mode of the representation of the error, and write it in the simplified form:

$$\varepsilon_j^n = A_j^n e^{ikj\Delta x}$$

Q1c

Write a Matlab function that updates q one time step using the BTCS scheme. Your function header should read as:

```
function [qnew] = BTCS_adv(qold,dt,dx,u)
```

where the inputs and outputs to the function are:

- dt is the timestep
- dx is the grid spacing
- u is the value of u in Eq 1
- qold is the vector of q values at the current step
- qnew is the vector of new q values calculated in the function.

Your function must implement the following boundary conditions:

- Assumes the value of q is ZERO on any boundary where the velocity u advects (carries) q INTO the domain (this is called an inflow boundary),
- Calculates the value of q on any boundary where the velocity u is advecting (carrying) q OUT of the domain (this is called an outflow boundary) by assuming $\frac{\partial q}{\partial x} = 0$. ***Use centred differences*** for this BC.
- Note that which boundary is inflow and which is outflow depends on whether u is > 0 or < 0.

Hint: You may use the Thomas algorithm or an in-built MATLAB function to solve the system of linear equations. Don't forget you need to modify the matrix for the boundary conditions.

Q1d

Modify the template **Lab06_Q1.m** so that it

1. Uses a domain that is defined on [0 ≤ x ≤ 10] with initial condition (at t = 0) given by

$$q(x,0) = e^{-\gamma(x-x_c)^2}$$

2. Uses the BCs specified in **Q1c**.
3. *Discretises the domain* with 250 intervals (251 points)
4. *Sets* u = 0.05, γ = 10 and x_c = 2
5. Integrates forward from t = 0 to t = 100 with the following CFL numbers: 0.2, 1.0, 1.1 and 5.

Create one figure for each value of the CFL number (i.e. 4 figures in total). On each figure, plot your predicted solution (i.e. $q(x,t)$ vs x) at approximately every 20 time units (i.e. at t = 0, 20, 40, 60, 80 and 100). These times can be "approximate" to allow for plot times being non-integer multiples of the timestep (e.g. 20.1 is close enough to 20).

On each of these four figures, compare your computed solution to the exact[1] solution (at the time at which you have plotted your calculated solution). Plot each **calculated curve with one colour line** and each **exact curve with one different colour line** (e.g. 6 green lines for calculated, and 6 magenta lines for exact). Make sure the legend clearly states which line is analytic and which is calculated. You will have 12 curves on each figure.

Q1e

In the same **Lab06_Q1.m** template, ***write a few sentences*** to the command window that

1. describes the key features of what you observe for each CFL number
2. relates these results back to your stability analyses in Questions 1b.
3. compares the results for different CFL numbers, i.e. what happens to the advected quantity q at different CFL.
4. explains why CFL = 1 did not manage to produce results that are comparable to the exact solution.

QUESTION 2 [20 MARKS TOTAL]

Background

An alternative way to discretise the spatial derivative of the advection equation (Eq 1) is to use a scheme that is known as "upwind differencing". It is defined as follows:

$$\left.\frac{\partial q}{\partial x}\right|_j^n = \begin{cases} \dfrac{q_j^n - q_{j-1}^n}{\Delta x}, & if\ u \geq 0 \\ \dfrac{q_{j+1}^n - q_j^n}{\Delta x}, & if\ u < 0 \end{cases} \qquad \text{(Eq 2)}$$

[1] If $q(x,0) = f(x)$ is the initial condition, then the exact solution at time = t is $q(x,t) = f(x\text{-}ut)$

Instead of estimating the derivative using centered differences, the upwind scheme estimates it using forward or backward differencing *in space* depending on the direction in which the flow is moving (i.e. whether u pis ositive or negative).

Q2a

Write the forward in time, upwind in space (FTUS) difference equation that approximates Eqn 1 when $\boldsymbol{u > 0}$.

Q2b

Undertake a von Neumann stability analysis for the FTUS scheme in Q2a and determine its stability criteria. Consider just ONE Fourier mode of the representation of the error, and write it in the simplified form:

$$\varepsilon_j^n = A_j^n e^{ikj\Delta x}$$

Q2c

Write a MATLAB function that updates the values of $q(x)$ by ONE TIMESTEP using the FTUS scheme. Ensure that your function can handle positive and negative values of u, (as specified in Eq 2) and has the function header

```
function [q_new] = FTUS(q,dt,dx,u)
```

Implement boundary conditions in the same way as in Q1.

NOTE: We can calculate the equation at an outflow boundary using the FTUS scheme because we do not need the value of the unknown outside the domain, because this is "down-wind" of the boundary, and down-wind points are never used in this scheme.

In the function, you need to check which direction the flow is coming from and implement appropriate BCs for the 2 cases.

Q2d

Modify the template **Lab06_Q2.m** and repeat **Q1d** using the FTUS scheme with the same time-steps.

Q2e

In the same **Lab06_Q2.m** template, w***rite a few sentences*** that describes

1. describes the key features of what you observe for each CFL number
2. relates these results back to your stability analyses in Questions 2c.
3. compares the results for different CFL numbers, i.e. what happens to the advected quantity q at different CFL.

Write a few sentences to the command window that comments on the statement

> *Using a smaller time step will always give you a better solution when solving a time-dependent numerical problem*

QUESTION 3 [15 MARKS TOTAL]

Background

The 2D heat equation expressed in Cartesian coordinates is given by

$$\frac{\partial T}{\partial t} = \alpha\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \qquad \text{Eqn 3}$$

where T is temperature and $\alpha > 0$ is the thermal diffusion coefficient.

Q3a

Write the BTCS scheme for the 2D heat equation in Eq 3 at node (j,k) for time "n+1". You **must move** any unknowns (at time "n+1") to the LHS and all known values (at time "n") to the RHS. The index j represents location in the x-direction, while the index k represents location in the y-direction.

Q3b

Derive the stability restriction on the timestep for the BTCS scheme in Q3a with thermal diffusion coefficient α. You will need to write the error as

$$\varepsilon_{jk} = A^n e^{iRj\Delta x} e^{iSk\Delta y}$$

where *R* and *S* are wavenumbers in the *x* and *y* directions, respectively. You can treat *R* and *S* in the same way we treat *k* in the 1D heat equation. Is the BTCS scheme unconditionally unstable, conditionally stable or unconditionally stable? Justify your answer.

Q3c

By hand, write the BTCS scheme in Q3a into a matrix problem of the form

$$\mathbf{T}^{n+1} = [\mathbf{A}]^{-1}\left([\mathbf{B}]\mathbf{T}^{n} + \mathbf{BC}\right)$$

where $\mathbf{T}$ is the temperature vector, **[A], [B]** are matrices that depend on the identity matrix [**I**], the Laplacian matrix [**L**], the timestep δt and the diffusion coefficient α. **BC** is a vector that stores values associated with the boundary conditions. Specify exactly what **[A], [B]** are equal to in terms of **[I], [L],** δt and α.

QUESTION 4 [40 MARKS TOTAL]

Background

In this question, you will be solving the 2D heat equation defined in Eqn 3 in a rectangular domain of -1 ≤ x ≤ 1 and -0.5 ≤ y ≤ 0.5. The domain has an initial temperature of zero and the boundary conditions are given by

$$T(x,y) = \exp(1 - x^2 - y^2)\sin(x^2 + y^2) \qquad \text{Eqn. 4}$$

You are asked to discretize the rectangular domain into Nx grid points in the x-direction and Ny grid points in the y-direction. Note that the first and last points in both the x- and y-directions are boundary conditions, which means that there will be a total of (Nx-2) x (Ny-2) unknowns.

Q4a

In your PDF for Question 1, ***outline the algorithm*** needed to solve Eqn 3 using the BTCS scheme from time = 0 to time = t_n, with a constant timestep, Δt (assume that $t_n = N t\Delta t$). Include the major steps required to set the problem up (don't include minor steps like setting grid spacing, etc.) This algorithm must be in the form of **pseudo code** or a **flow chart**. Example of a pseudo code for solving an ODE using the Euler method is shown in the snippet below.

```
Set up parameters of the problem
Determine stepsize and number of steps
for each time step
    Solve the Euler method
    Store calculated values in vector
end
```

Q4b

Write a Matlab function that sets up the Laplacian matrix **L** and the boundary condition vector **BC**. Your function header must be written as

```
function [L,BC] = MatSetup(Nx,Ny,xg,yg,alpha,BCf)
```

where

- Nx and Ny are the number of points in the x- and y-directions, respectively
- xg and yg are column vectors containing the grid points of x and y, respectively.
- alpha is α, i.e. the thermal diffusion coefficient
- BCf is function describing the boundary condition (see Eqn 4)
- L is the ((Nx-2)*(Ny-2) x (Nx-2)*(Ny-2)) Laplacian matrix
- BC is ((Nx-2)*(Ny-2) x 1) column vector on the RHS that accounts for the boundary conditions

NOTE: You can define an anonymous function that converts a 2D node number (j,k) = ($j\delta x$, $k\delta y$) to a global node number for the unknowns. Your function will look like

nn=@(j,k,Nx) {*some function of j, k, Nx*};

(See Workshops 25-27 for examples of what this looks like). When setting up the matrix problem the function nn (or at least a similar process) can be used to reliably determine the row and column number of each entry in the Laplacian matrix and the row number in the BC matrix.

Q4c

Modify **Lab06_Q4.m** to solve the 2D Heat equation using the BTCS scheme (i.e. Implement your algorithm from Q4a) together with the function you have written in Q4b. Set α = 0.1 and time step Δt = 1 s, and solve the PDE from t = 0 to t = 50 s for (Nx, Ny) = (21, 11), (41, 21), (61, 31) and (81, 41).

Using the command **'tic'** and **'toc'**, ***determine how much time*** is spent setting the matrix problem up, i.e. the time spent assembling the Laplacian matrix versus how much time is spent undertaking the time-stepping solution. (**NOTE: The time taken for plotting should not be included as part of the time taken for time-stepping).**

Print out the times for each (Nx, Ny) grid and **write a few sentences** comparing the times and explain what they mean.

In separate figures, ***plot contours*** of your solution ***including*** the boundary values for each (Nx, Ny). You will have to **reshape your 1D vector** of unknowns **into a 2D matrix** and embed that into a slightly bigger 2D matrix and add boundary conditions.

Q4d

Repeat Q4c with timestep of Δt = 50 s (Just do this inside the same script file **(Lab06_Q4.m)** with a loop for the 2 different time steps).

Is the solution stable or unstable in this case? Should it be? ***Write a comment*** to the command window that compares your solution calculated with one large time step to that obtained in 50 smaller time steps.

NOTE: the following MATLAB code will contour a 2D grid of data values stored in Qplot, where xg is a 1-D vector that specifies the grid x-locations (length=Nx) and yg is a 1-D vector that specifies the grid y-locations (length=Ny).

```
figure
clev=linspace(0.25,1,25);  % Set the desired contour levels
contourf(xg,yg,Tplot',clev);
```

Poor Programming Practices **[-10 Marks]**

(Includes, but is not limited to, poor coding style or insufficient comments or unlabeled figures, etc.)

(END OF LAB)