Assignment 2

Important

Marks: This assignment is worth **35%** of the overall assessment for this course.

Due Date and time: Monday, 26 October 2020, 11:59pm, via submit on Canvas. You will lose penalty marks at the rate of **10% per day** or part day late, and the assignment will not be marked if it is more than **5 days** late (counted from the due date/time).

Learning outcomes

In this assignment you will demonstrate your understanding of solving engineering problems using numerical computations and assessing particular algorithms. The objectives of this assignment are to program algorithms for root-finding, solving systems of linear algebraic equations, performing least-squares approximations and interpolations, regressing solutions and solving a differential equation using different integration and differentiation schemes.

1 Root finding [9/35 Marks]

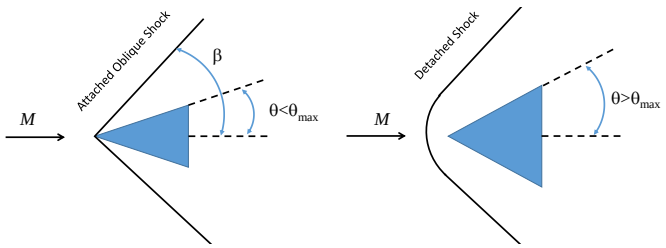

Figure 1: Difference in shock wave type for different wedge angles.

Imagine a wedge-shaped object flying through air at supersonic speeds (see Fig. 1). In compressible flow theory (covered in MCEN90008 Fluid Dynamics), it is known that an oblique shock wave forms at the front tip of this object under certain conditions. The equation relating the wedge half-angle θ to the shock oblique angle β and the Mach number M is given by

$$\tan(\theta) = 2\cot(\beta)\frac{M^2\sin^2(\beta) - 1}{M^2(\gamma + \cos(2\beta)) + 2}\ , \tag{1}$$

where $\gamma = 1.4$ is the ratio of specific heats. For a given M, as you keep increasing θ, there is a critical angle, θ_{max} where the shock becomes detached. We can also recast Eq. (1) into the following form

$$f(\beta) = 2\cot(\beta)\frac{M^2\sin^2(\beta) - 1}{M^2(\gamma + \cos(2\beta)) + 2} - \tan(\theta)\ . \tag{2}$$

Your tasks are the following:

Analytical solution for $\theta = 0°$

For $\theta = 0$, i.e. the object would be a flat plate, show that two possible solutions for β are

$$\beta_L = \arcsin\left(\frac{1}{M}\right)\ , \qquad \beta_U = 90°\ . \tag{3}$$

β_U and β_L are usually called the strong shock and weak shock solutions, respectively. Even though we can mathematically obtain two possible solutions, in reality, or physically, only the weak shock solution occurs. Note also that in order for the solution to be physically realisable, $\theta < \beta < 90°$.

Graphical solution

Plot $f(\beta)$ vs β for

1. $M = 2.0$ and $\theta = \{3, 16, 26, 37, 44\}$ degrees
2. $M = 4.5$ and $\theta = \{3, 16, 26, 37, 44\}$ degrees

Describe how β_U and β_L change with θ and M. Can you identify from your plots the approximate value of θ_{max}?
Note: Don't forget to change the angle from degrees to radians when you use sinusoidal functions in your computer program.

C program to solve shock-wave equation

Write a C code that solves Eq. (2) using the Newton–Raphson method to find the root of $f(\beta)$, regarding θ and M as parameters and solving for β.

(a) Write your C program such that it uses the Newton–Raphson method to solve $f(\beta) = 0$. What values of β_L and β_U do you think might be appropriate to use as an initial guess? For $M = 4.5$ and $\theta = 26°$, you should find that $\beta_L = 38.17378...°$ and $\beta_U = 81.9852...°$

(b) Use your computer program to solve $f(\beta) = 0$ (i.e. find β_U and β_L values) for $M = 2.5$, 3.5, 4.5, 5.5, 6.5, 7.5. Remember that for $\theta = 0$, $\beta_L = \arcsin\left(\frac{1}{M}\right)$ and $\beta_U = 90°$. Plot β vs θ for all the M's. That is, plot values of θ on the horizontal axis and corresponding values of β on the vertical axis. This plot is called the $\theta - \beta - M$ diagram and you will find it very useful if you decide to do MCEN90008 Fluid Dynamics in the future. Your solution to this part of the assignment should look like Fig. 2 which shows the solution for $M = 5$. You can plot your results obtained from your C code with your program of choice, e.g. Matlab, Excel, etc.

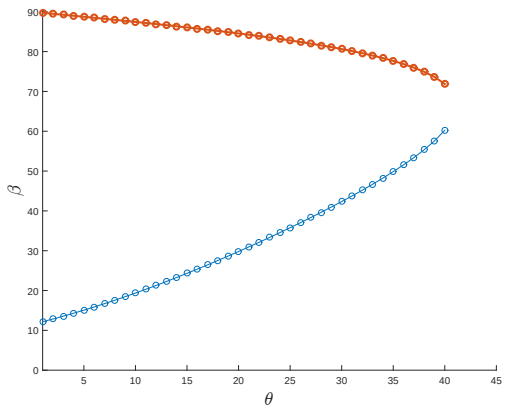

Figure 2: $\theta - \beta - M$ diagram for $M = 5.0$.

The implementation for this task is to be done in the function named **`shockwave()`**. The input parameters are to be read in from the input file **`in_shock.csv`**. The input file consists of two parts, as shown below:

```
M,theta,beta_l,beta_u,gamma
4.5,26.0,0.0,0.0,1.4
M
2.5
3.5
4.5
5.5
6.5
7.5
```

The first two lines correspond to the part (a) where for M = 4.5 and $\theta = 26^0$, you need to compute the values of β_L and β_U. You will notice the initial values set here are 0^0 apiece so that you can modify them to find the right initial guess to compute the angles. The next set of lines corresponds to part (b) where you will evaluate *for different M*, the values of β_L and β_U for different values of θ (ranging from 0^0 up to θ_{max} at an increment of 1^0). Outputting the results of this task are to be done only for part (b) into a file named **`out_shock.csv`** in the format as shown below. This example only shows part of the results for $\theta = 0^0, 1^0$ for the set of Mach numbers chosen (you can use this to validate your code). The output of M, θ, β_L and β_U is to be done up to **4 decimal places**:

```
M,theta,beta_lower,beta_upper
.
.
3.5000,0.0000,16.6016,90.0000
3.5000,1.0000,17.2674,89.6933
.
.
4.5000,0.0000,12.8396,90.0000
4.5000,1.0000,13.4859,89.7376
.
.
```

You must dynamically allocate space for the Mach numbers so that your implementation may work for a different set of M values (we will evaluate your code for a different set of M values!). For each Mach number, upon increasing θ by 1^0, you will reach the maximum θ beyond which the solutions of β are not physically relevant. You will write to file only up to this maximum θ per Mach number.

2 Regression [4/35 marks]

Here, an alternative way of solving a Least Squares Problem is considered. Recall from the lecture that

$$\begin{bmatrix} \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & N \end{bmatrix} \begin{Bmatrix} a \\ b \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{N} x_i y_i \\ \sum_{i=1}^{N} y_i \end{Bmatrix} \tag{4}$$

is the system of equations for linear regression $\hat{y} = ax + b$.

Show that:

$$b = \overline{y} - a\overline{x}, \quad a = \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{5}$$

where $\overline{x} = \sum_{i=1}^{N} x_i/N$ is the mean of x_i.

By considering Eq. 5, when will linear regression *fail* to find a solution?

3 Linear Algebraic Systems [5/35 marks]

Consider the following tri-diagonal system:

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & \dots & 0 & 0 & a_N & b_N \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{Bmatrix} = \begin{Bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ \vdots \\ r_N \end{Bmatrix} \tag{6}$$

(a) Using Gauss elimination, show that the above matrix can be rewritten as

$$\begin{bmatrix} b_1^* & c_1 & 0 & 0 & \dots & 0 \\ 0 & b_2^* & c_2 & 0 & \dots & 0 \\ 0 & 0 & b_3^* & c_3 & \dots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & b_{N-1}^* & c_{N-1} \\ 0 & \dots & 0 & 0 & 0 & b_N^* \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{Bmatrix} = \begin{Bmatrix} r_1^* \\ r_2^* \\ r_3^* \\ \vdots \\ \vdots \\ r_N^* \end{Bmatrix} \tag{7}$$

where

$$b_i^* = \begin{cases} b_i & \text{for} \quad i = 1 \\ b_i - a_i c_{i-1}/b_{i-1}^* & \text{for} \quad i = 2, 3, ..., N \end{cases}$$

and

$$r_i^* = \begin{cases} r_i & \text{for} \quad i = 1 \\ r_i - a_i r_{i-1}^*/b_{i-1}^* & \text{for} \quad i = 2, 3, ..., N \end{cases}$$

Thus the solution to the original tri-diagonal matrix can be written as

$$x_i = \begin{cases} r_i^*/b_i^* & \text{for} \quad i = N \\ (r_i^* - c_i x_{i+1})/b_i^* & \text{for} \quad i = N-1, N-2, ..., 1 \end{cases}$$

The algorithm outlined above is called the *Thomas algorithm*. When applicable, it is a very efficient method for solving linear tri-diagonal systems.

(b) Write a C code using the Thomas algorithm to solve the tri-diagonal system shown in Eq. 6. Since the tri-diagonal system is a banded matrix, you need not store all the zeros! Instead, your code should take as an input the vectors a_i, b_i, c_i and r_i. The output from your C code should be the solution vector x_i. Validate your code by solving the following tri-diagonal system

$$\begin{bmatrix} 1 & -2 & 0 & 0 & 0 \\ 2 & 4 & 5 & 0 & 0 \\ 0 & 8 & -9 & 2 & 0 \\ 0 & 0 & 6 & 3 & 4 \\ 0 & 0 & 0 & 6 & 3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ -3 \\ 4 \\ 5 \end{Bmatrix} \tag{8}$$

and ensuring that you obtain the expected solution

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{Bmatrix} = \begin{Bmatrix} 0.561782 \\ -0.219109 \\ 0.350575 \\ 0.954023 \\ -0.241379 \end{Bmatrix} . \tag{9}$$

You will implement the code for this task in the function **`linsolve()`**, where you will read in as an input the vectors a_i, b_i, c_i and r_i from the file **`in_linsolve.csv`** in the following format:

```
a,b,c,r
0.000000,1.000000,-2.000000,1.000000
2.000000,4.000000,5.000000,2.000000
8.000000,-9.000000,2.000000,-3.000000
6.000000,3.000000,4.000000,4.000000
6.000000,3.000000,0.000000,5.000000
```

The output from your implementation should be the solution vector x_i, written to **`out_linsolve.csv`** in the following format (up to 6 decimal places):

```
x
0.561782
-0.219109
0.350575
0.954023
-0.241379
```

Your implementation should allocate dynamically the space for the values of a_i, b_i, c_i and r_i such that your code would work for different problem sizes as well (We will actually test your code for a different problem with different size).

4 Interpolation [4/35 marks]

For this task, you will perform a cubic spline interpolation on the data provided in the file **in_interp.csv**. First, plot the data (using MATLAB or Excel) in order to get an idea of the behaviour $f(x)$ should have. Next, write C code that uses cubic splines in order to estimate the function $f(x)$ that cuts through the data. You will then use your code to calculate the value(s) of $f(x)$ at $x_o = 3.5$, where x_o is a runtime input parameter. You will implement your method in **interp()**, reading the values of x and $f(x)$ from the file **in_interp.csv** in the following format:

```
x,f(x)
4.440706476030278,4.403874813710879
4.116904962153069,4.403874813710879
.
.
.
```

and outputting the values of the interpolated value to **out_interp.csv** in the following format (up to 6 decimal places).

```
xo,f(xo)
3.500000,0.234655
3.500000,1.107865
```

The output provided above is an example and does not constitute the solution. There may be more than 1 solution possible here and you must be able to identify the correct interval(s) to compute the interpolated value. For the C implementation, you must dynamically allocate space for x, $f(x)$ and write your code such that it can work for a different set of input data and different x_o value. Finally, you must plot the interpolated function using either MATLAB or Excel. How does the interpolated function look compared with the actual datapoints?

5 Differentiation, differential equations [13/35 marks]

The one-dimensional transport of a quantity $\phi(x,t)$ in a fluid moving at a constant velocity u can be described by the following linear advection equation:

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = 0\ . \tag{10}$$

(a) Write a C program that solves (10) on the interval $0 \leq x \leq 1$, and for times $0 \leq t \leq t_{final}$, using the velocity $u = 1$. The spatial interval is to be discretized by $N_x + 1$ points. That is $x_i = i\Delta x$ with $i = 0, 1, 2, ..., N_x$, where the equidistant grid spacing $\Delta x = 1/N_x$.

Equation (10) can also be written as

$$\frac{\partial \phi}{\partial t} = \text{RHS}(\phi) \tag{11}$$

where RHS denotes the 'right-hand-side' of the equation containing the spatial derivative $-u\partial\phi/\partial x$.

Discretize the advection equation (10) by using the so-called "leapfrog" scheme (central difference approximation for both the spatial and temporal derivatives):

$$\frac{\phi^{n+1} - \phi^{n-1}}{2\Delta t} \approx \text{RHS}(\phi^n) \text{ for } \quad n \geq 2 \tag{12}$$

and

$$\text{RHS}(\phi_i) \approx -u\frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} \text{ for } \quad i = 1, ..., N_x - 1 \tag{13}$$

where n is the time index so that for instance ϕ_i^{n+1} is ϕ value at time level $n+1$ and at point x_i. Δt is the numerical time step for the time integration. Note that in the leapfrog scheme, both ϕ^n

and ϕ^{n-1} are required to update ϕ^{n+1}. Thus, the scheme does not work for the first time iteration ($n = 1$) and instead the forward Euler scheme is to be used:

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} \approx \text{RHS}(\phi^n) \text{ for } \quad n = 1 \tag{14}$$

The so-called "periodic" boundary condition is to be adopted at the end points. That is, when evaluating the spatial derivatives at $i = 0$ and $i = N_x$, equation (13) becomes:

$$\text{RHS}(\phi_i) = -u\frac{\phi_1 - \phi_{N_x-1}}{2\Delta x} \quad \text{for} \quad i = 0, N_x \tag{15}$$

The following initial condition for ϕ shall be used to start your simulation:

$$\begin{aligned} 0 \le x < 0.125 \quad &\rightarrow \quad \phi(x, t = 0) = 0 \\ 0.125 \le x \le 0.375 \quad &\rightarrow \quad \phi(x, t = 0) = 0.5\Big(1 - \cos\Big(8\pi(x - 0.125)\Big)\Big) \\ 0.375 < x \le 1 \quad &\rightarrow \quad \phi(x, t = 0) = 0\,. \end{aligned}$$

For your code to run stably (i.e. the solution does not "blow up" towards infinity), your time step needs to satisfy the so-called CFL condition:

$$\text{CFL} = u\frac{\Delta t}{\Delta x} \le 1\,. \tag{16}$$

You will implement your routines in the function **`advection()`**, reading in the values of u, N_x, CFL and t_{final} from **`in_advection.csv`** in the following format:

```
u,Nx,CFL,t_final
1.0,200,0.50,1.0
```

The parameter t_{final} determines the total number of time iterations required (N_t). That is, your program will integrate the advection equation up to time $t_{final} = N_t\Delta t$. When computing N_t for a given t_{final}, round N_t to next higher integer in case $t_{final}/\Delta t$ is not a full integer. Finally, your program shall output the computed solution of ϕ at the end of the temporal iteration (i.e. at time $t = N_t\Delta t$) to the file **`out_advection.csv`** in the following format (up to 6 decimal places).

```
x,phi
0.000000,0.000000
0.010000,0.000394
0.020000,0.000907
.
.
.
```

(b) Run your code until time $t_{final} = 1.0$ for the following cases:

- Chose two different resolutions Δx, by setting N_x to 100 and 200.
- For each N_x, chose the time step Δt from the CFL condition, using CFL=1, 0.8 and 0.25.

Plot your solutions at $t = 1.0$ (six plots in total) and compare with the exact solution. The exact solution is the same as the initial condition but shifted by $u \cdot t$ in the positive x direction. That is, if the initial condition is given by the function $g(x)$, then the exact solution at time t would be $g(\text{mod}(x - u \cdot t, 1))$ where $\text{mod}(a, b)$ is the modulus function which computes the remainder of a divided by b. How well does the solution agree with the exact solution? Discuss:

- How does the agreement of the numerical prediction with the exact solution depend on the grid resolution Δx?
- How does the agreement of the numerical prediction with the exact solution depend on the CFL number?
- What happens if you chose CFL>1, for example CFL=1.002?

Submission

This assignment, unlike assignment 1, consists of two parts

1. A project report, detailing any derivations and solutions and displaying the required graphs
2. C programs developed to solve some of the problems

Project Report

Your project report need not be a full technical report but should state all approximations made and use figures of results to back up any conclusions. Be sure to include enough detail so that your results could be reproduced by another researcher (or yourself at a future date!) wishing to check or extend your findings. Your report will be primarily assessed on the completeness of the results, and the visual/logical effectiveness of the manner in which they are presented. **Please type your report** - scanned handwritten notes are sometimes too difficult to read and mark and therefore will not be accepted.

C programs

The following files are provided to you for this assignment:

- **`main.c`**, where the parsing of command line arguments is to be done and timing for each task implemented.
- **`tasks.c`**, where you will implement four functions **`shockwave()`**, **`linsolve()`**, **`interp()`** and **`advection()`**, one for each task.
- **`tasks.h`**, which acts as a header file to link the C file to the main file. You may edit this to add any **`struct`**'s or the input arguments to the functions.
- Input files **`in_shockwave.csv`**, **`in_linsolve.csv`**, **`in_interp.csv`** and **`in_advection.csv`** which you must use as input during execution of your program. During your submission, your code will be tested on different input files from the ones you were provided. Please make sure any data structures used to read in these files are dynamically allocated to avoid any errors during runtime.

You are free to use and adopt guide programs provided during the lectures and workshop exercises. Remember to fill in your details in each file you write, such as the student name and student number.

Submission on Canvas LMS

You will need to submit your work in Canvas LMS. You can submit partially completed tasks as soon as you have them. You can submit multiple times but only the latest submission will be marked. Try not to wait until last minute to make your submission. You will be allowed to submit after the due date. But all submissions after the due date will be marked as late submissions and the late submission policy will be applied in grading your assignment.

Your files need to be packed into a single zip file which will contain **`main.c`**, **`tasks.c`**, **`tasks.h`** and your project report **in PDF file format**. Do not include the input or output CSV files in your submission. You must name your file using your SIS Login ID. For example if your username is DaRodriguez, your zip file will be named **`DaRodriguez.zip`**. If the grader fails to extract the required files from the zip file you will get zero marks. Submitting individual files in Canvas will not be possible. Additionally, failure to comply with the above naming format will infer a penalty of 10% of your marks.

Your code will be checked for compile and run time errors in the server **`dimefox`**. Make sure that your code compiles and runs successfully in the server to avoid losing marks. That is, transfer you codes and the input CSV files to **`dimefox`** and compile and run your code to see if it works without errors. Once you are satisfied with your results, you can submit your work through Canvas as mentioned earlier.

To compile your code on **`dimefox`**:


```
gcc -Wall -std=c99 main.c tasks.c -o exec -lm
```

You can then test your compiled code using:

```
./exec in_shockwave.csv in_linsolve.csv in_interp.csv 3.5 in_advection.csv
```

There are 5 command line arguments here, one for each of the 4 functions. The argument number 4 (i.e. the value **`3.5`**) is part of the interpolation task. It corresponds to the value of x_o at which the interpolated value is to be outputted. Once your code works for the provided input files, it is recommended to check your code for different input files and different x_0 values and see if your code still works and outputs expected results. If this works, your code is now submission ready.

Points to consider

- Remember to fill in your details in each file you write, such as your name and ID.
- For the purposes of the report, you can output as many files or screen outputs as you need. These outputs can be used to generate graphs/plots and values for the different tasks. Once you have all information you need for the report, your code must be made submission worthy i.e. only output the files described above (4 output files are expected). Your code must not expect user input once you execute it, all required inputs would come from the input files or the terminal before execution.
- You have to parse the command line arguments (all input file names and any command line values) i.e. do not hard-code the names of the files or the value for interpolation task. This is because we will be using our own input files, with different filenames and different locations when assessing your work.
- Plan before you write your code. Consider all possibilities regarding different input sizes. Dynamically allocating structures for the input file contents is very important so that for files with more or less entries don't end up giving you errors during execution.
- You should not use functions outside of the C standard library and maths library.
- Submissions that can not be compiled or run in **`dimefox`** will receive **zero marks for the programming part**.
- Submissions are also limited to a maximum runtime of 200 seconds and maximum file size per task of 2 MB. Failing to comply with these limits would give errors during testing of your code.
- Since each task can be assessed individually, you can submit your code with just one or two tasks implemented. The feedback will accordingly skip over tasks which are not implemented and only look at the completed tasks.
- Only your last submission is stored in Canvas i.e. everytime you resubmit, the new submission overwrites the previous version. Keep a backup of your previous versions somewhere safe in case your latest submission works worse than the previous.

6 Getting Help

There are several ways for you to seek help with this assignment. First, go through all the instructions provided in this document in detail. It is likely that your question has been answered already. Second, you may also discuss the assignment on the **Discussion Assignment 2** in Canvas. However, please **do not** post any source code on the discussion board. Finally, you may also ask questions during the workshops. You can also contact either the lecturer Aman Kidanemariam (aman.kidanemariam@unimelb.edu.au) or the head tutor David Rodriguez (david.rodriguezSanchez@unimelb.edu.au) directly for unresolved issues. Students seeking extensions for medical or other "outside my control" reasons should email the lecturer as soon as possible after those circumstances arise.