ENGR30003: Numerical Programming for Engineers
Semester 2, 2019 / Assignment 2

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

Due Date : **Monday, 21st October 2019, 5:00pm**, via `dimefox` submission. You will lose penalty marks at the rate of 10% per day or part day late, and the assignment will no be marked if it is more than 5 days late.

1 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.

2 Rootfinding [9/35 marks]

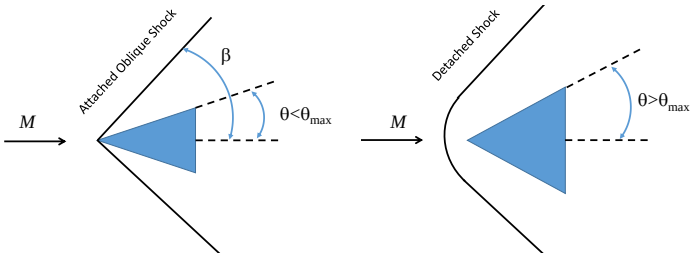

Figure 1: Difference in shock wave type for different wedge angles.

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:

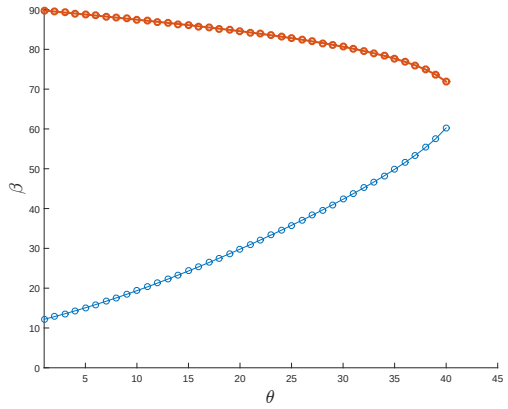

Figure 2: $\theta - \beta - M$ diagram for $M = 5.0$.

2.1 Analytical solution for $\theta = 0°$ [1 mark]

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

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

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, $\theta < \beta < 90°$.

2.2 Graphical solution [2 mark]

Plot $f(\beta)$ vs β for

a. $M = 1.5$ and $\theta = 5°$, $10°$ and $15°$

b. $M = 5.0$ and $\theta = 20°$, $30°$ and $45°$

Indicate how β_U and β_L change with θ and M. Can you identify from your plots the approximate value of θ_{max}?

PLEASE REMEMBER to change the angle from degrees to radians when you use sinusoidal functions in your computer program.

2.3 C program to solve shock-wave equation [6 marks]

Your task is to 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 $M = 5.0$ and $\theta = 20°$, you should find that

$$\beta_L = 29.80092...^\circ \,, \quad \text{and} \quad \beta_U = 84.55625...^\circ$$

(b) Extend your C program to find β_U and β_L values for $M = 5.0$ and $0 \le \theta \le \theta_{max}$. Remember that for $\theta = 0$, $\beta_L = \arcsin\left(\frac{1}{M}\right)$ and $\beta_U = 90°$. Plot values of θ on the horizontal axis and corresponding values of β on the vertical axis. Your solution to this part of the assignment should look like Fig. 2. Note that you can plot your results obtained from your C code with your program of choice, e.g. Matlab, Excel, etc.

(c) Use your computer program to solve $f(\beta) = 0$ for M = 2.0, 3.0, 4.0, 6.0, 7.0, 8.0. Plot β vs θ for all the M's. 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.

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

```
M, theta , beta_l , beta_u ,gamma
 5.0 ,20.0 ,0.0 ,0.0 ,1.4
M
 5.0
M
 2.0
 3.0
 4.0
 6.0
 7.0
 8.0
```

The first two lines corresponds to the part (a) where for M = 5.0 and $\theta = 20^0$, 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 line corresponds to part (b) where you will for $M = 5$ evaluate the values of β_L and β_U for different values of θ (increments of 1^0 from 0^0 up to θ_{max}). The next set of lines corresponds to part (c) where you will evaluate *for different M*, the values of β_L and β_U for different values of θ (increments of 1^0 from 0^0 up to θ_{max}). Outputting the results of this task are to be done only for part (c) into `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, β_U, β_L is to be done up to 6 decimal places while θ as an integer:

```
M, theta , beta_lower , beta_upper
.
.
3.000000,0,19.471221,90.000000
3.000000,1,20.157561,89.649912
.
.
4.000000,0,14.477513,90.000000
4.000000,1,15.131265,89.719941
.
.
```

You must dynamically allocate space for the Mach numbers so that your implementation may work for a different set of M. 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.

3 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:

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

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

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

4 Linear Algebraic Systems [5/35 marks]

Consider the following tri-diagonal system:

$$\begin{bmatrix} a_1 & b_1 & 0 & 0 & \dots & 0 \\ c_2 & a_2 & b_2 & 0 & \dots & 0 \\ 0 & c_3 & a_3 & b_3 & \dots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c_{N-1} & a_{N-1} & b_{N-1} \\ 0 & \dots & 0 & 0 & c_N & a_N \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{Bmatrix} = \begin{Bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ \vdots \\ \vdots \\ Q_N \end{Bmatrix} \tag{6}$$

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

$$\begin{bmatrix} a_1^* & b_1 & 0 & 0 & \dots & 0 \\ 0 & a_2^* & b_2 & 0 & \dots & 0 \\ 0 & 0 & a_3^* & b_3 & \dots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & a_{N-1}^* & b_{N-1} \\ 0 & \dots & 0 & 0 & 0 & a_N^* \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{Bmatrix} = \begin{Bmatrix} Q_1^* \\ Q_2^* \\ Q_3^* \\ \vdots \\ \vdots \\ Q_N^* \end{Bmatrix} \tag{7}$$

where

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

and

$$Q_i^* = \begin{cases} Q_i & \text{for} \quad i = 1 \\ Q_i - c_i Q_{i-1}^*/a_{i-1}^* & \text{for} \quad i = 2,3,...,N \end{cases}$$

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

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

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

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 Q_i. The output from your C code should be the solution vector x_i.

Use your code to solve the following tri-diagonal system

$$\begin{bmatrix} 1 & -2 & 0 & 0 & 0 & 0 \\ 2 & 4 & 5 & 0 & 0 & 0 \\ 0 & 8 & -9 & 2 & 0 & 0 \\ 0 & 0 & 6 & 3 & 4 & 0 \\ 0 & 0 & 0 & 6 & 3 & 2 \\ 0 & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ -3 \\ 4 \\ 5 \\ 1 \end{Bmatrix} \tag{8}$$

You will implement the code for this task in the function `linalgbsys()`, where you will read in as an input the vectors a_i, b_i, c_i and Q_i from the file `in_linalsys.csv`. The output from your implementation should be the solution vector x_i, written out to `out_linalsys.csv` (up to 6 decimal places). Your implementation should allocate dynamically the space for the values of a_i, b_i, c_i and Q_i such that your implementation would work for different problem sizes as well.

5 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 $x_o = 4$, where x_o is to be input at runtime from stdout. You will implement your method in `interp()`, reading the values of x and $f(x)$ from the file `in_interp.csv` and outputting the values of the interpolated value to `out_interp.csv` in the following format (up to 6 decimal places).

```
xo , f(xo)
4.000000,0.234655
4.000000,1.107865
```

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. Finally, you must plot the interpolated function using either MATLAB or Excel. How does the interpolated function look compared with the actual datapoints?

6 Differentiation, differential equations [13/35 marks]

Write a C program that solves the modified wave equation

$$\frac{\partial f}{\partial t} + c\frac{\partial f}{\partial x} = 0 \tag{9}$$

on the interval $x = [0;1]$, and for times $0 \le t \le 0.2$, using the phase speed $c = 1$.

Most transport equations can also be written as

$$\frac{\partial f}{\partial t} = \mathrm{RHS}(f) \tag{10}$$

where RHS denotes the 'right-hand-side' of the equation to be solved, containing all spatial derivatives, in this case $-c\frac{\partial f}{\partial x}$.

The wave equation is to be solved on the spatial interval $0 \le x \le 1$, that is discretized using $N_x + 1$ points x_i, with $i = 0, 1, 2, ..., N_x$. The equidistant grid spacing therefore is $\Delta x = 1/N_x$.

In order to discretize the time derivative $\frac{\partial f}{\partial t}$, use the second-order accurate Runge–Kutta integration

$$\begin{aligned} f_i^{n+0.5} &= f_i^n + \Delta t \cdot \mathrm{RHS}(f_i^n) \\ f_i^{n+1} &= f_i^n + \frac{\Delta t}{2} \cdot \left[\mathrm{RHS}(f_i^n) + \mathrm{RHS}(f_i^{n+0.5})\right] , \end{aligned} \tag{11}$$

where n is the time index so that f_i^n is the function value at time level n at point x_i and f_i^{n+1} is the function value at time level $n+1$ at point x_i, superscript $n+0.5$ denotes an intermediate (time) level, and Δt is the numerical timestep for the time integration.

Two different finite-difference approximations for the spatial derivative are to be coded up:

- The first-order accurate upwind scheme:

$$\frac{\partial f}{\partial x} \approx \frac{f_i^n - f_{i-1}^n}{\Delta x} \quad \text{for} \quad i = 1, ..., N_x$$

  For $i = 0$, use the boundary stencil

$$\frac{\partial f}{\partial x} \approx \frac{f_{i+1}^n - f_i^n}{\Delta x} = \frac{f_1^n - f_0^n}{\Delta x}$$

- The second-order accurate central scheme:

$$\frac{\partial f}{\partial x} \approx \frac{f_{i+1}^n - f_{i-1}^n}{2\Delta x} \quad \text{for} \quad i = 1, ..., N_x - 1$$

  For $i = 0$, use the boundary stencil

$$\frac{\partial f}{\partial x} \approx \frac{f_{i+1}^n - f_i^n}{\Delta x} = \frac{f_1^n - f_0^n}{\Delta x}$$

  and for $i = N_x$ use the boundary stencil

$$\frac{\partial f}{\partial x} \approx \frac{f_i^n - f_{i-1}^n}{\Delta x} = \frac{f_{N_x}^n - f_{N_x-1}^n}{\Delta x}$$

The initial condition for f is

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

6.1 Suggested Code Structure

In the following possible steps for coding up the methods are suggested.

a. Set up main program that does the following

- Allocate arrays for the functions f_i^n, f_i^{n+1}, and $f_i^{n+0.5}$, etc, where $i = 0, 1, 2, ..., N_x$ (keep N_x a parameter so that you can change it).
- Write the initial condition into the array f_i^n, and write to a file for later visualization.
- Set up loop over the number of timesteps you want to run (each with Δt) - within this loop you want to call your RK2 routine.
- Write intermediate solutions to file for later visualization at several time instances.

b. Code up your RK2 routine:

- call a 'RHS' routine to get $\frac{\partial f}{\partial x}$, with the 'RHS' routine returning 'RHS(f_i^n)', i.e. the spatial derivatives are computed based on the function values at time level n
- compute $f_i^{n+0.5}$ as $f_i^{n+0.5} = f_i^n + \Delta t \text{ RHS}(f_i^n)$
- call 'RHS' routine again, now passing $f_i^{n+0.5}$ so that it returns 'RHS($f_i^{n+0.5}$)', i.e. the spatial derivatives are computed based on the function values at intermediate level $n + 0.5$
- compute $f_i^{n+1} = f_i^n + \frac{1}{2}\Delta t \left[\text{RHS}(f_i^n) + \text{RHS}(f_i^{n+0.5})\right]$

c. Code up your 'RHS' routine. In this routine, implement the two options for taking the spatial derivative, the upwinding scheme and the central scheme.

For your code to run stably, your time steps need to satisfy the so-called CFL condition, specifying that $\frac{c\Delta t}{\Delta x} \leq 1$.

You will implement your routines in the function `waveeqn()`, reading in the values of c, N_x and CFL from `in_waveeqn.csv`. This will allow you to compile your code just once and run it for different values of c, N_x and CFL by just changing the infile. For code assessment, you will write out the solutions from your two finite-difference approximations versions to `out_waveeqn_1U.csv` and `out_waveeqn_2C.csv`, respectively, the k^{th} time the time loop executes (this depends on what N_x, CFL are) in the following format (up to 6 decimal places).

```
x,f(x)
0.000000,0.000000
0.010000,0.000394
0.020000,0.000907
.
.
.
```

The k^{th} time will be read in from the infile from the parameter `out_iter`.

6.2 Tasks

Run your code until a time of $t = 0.2$ for both finite-difference approximations (1st-order upwind and 2nd order central) for the following cases:

- Chose two different resolutions Δx, by setting N_x to 80 or 200.
- For each resolution, chose the timestep Δt from the CFL number, using CFL=1, 0.75 and 0.25.

Output your results for the time-levels $t = 0.05, 0.1, 0.15, 0.2$ and plot the exact solution as well, which is the same as the initial condition, shifted by $c \cdot t$ in increasing x. How well does the solution at $t = 0.2$ 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 either method, for example CFL=1.002?

7 Submission

This assignment, unlike assignment 1, consists of two parts

a. A project report, detailing any derivations and solutions and displaying the required graphs

b. C programs developed to solve some of the problems

You need to submit your programs *and report* for assessment; **Submission of the report and the code will be done via** `dimefox`. You may discuss your assignment work during your workshop, and with others in the class, but what gets typed into your programs and the report must be **individual** work, not copied from anyone else. So, do **not** give a hard or soft copy of your work to anyone; do **not** "lend" your "Uni backup" memory stick to others for any reason at all; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "**no**" when they ask for a copy of, or to see, your programs or report, pointing out that your "**no**", and their acceptance of that decision, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode. Students whose programs or reports are so identified will be referred to the Student Center.* See `https://academichonesty.unimelb.edu.au` for more information.

7.1 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 (using appendices as necessary) 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.

7.2 C programs

The C codes are to be submitted on `dimefox` where they would be tested on different inputs from the ones you were provided to test your implementations.

7.2.1 Provided Code

- `main.c`, where the parsing of data from command line is to be done and timing for each task implemented.
- `tasks.c`, where you will implement four functions `shockwave()` (Question 2), `linalgbsys()` (Question 4), `interp()` (Question 5) and `waveeqn()` (Question 6), 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_shock.csv` (Question 2), `in_linalsys.csv` (Question 4), `in_interp.csv` (Question 5) and `in_waveeqn.csv` (Question 6) which you must use as input during execution of your program. During your submission, your code will be tested on different infiles from the ones you were provided. Please make sure any data structures used to read in these infiles are dynamically allocated to avoid any errors during runtime.

Remember to fill in your details in each file you write, such as the student name and student number. Key points about your code for this assignment you need to understand are as follows:

- For the purposes of the report, you can output as many files or terminal 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 outfiles described above (5 outfiles are expected). This means your code must not expect user input once you execute it, all inputs would come from the infiles or the terminal before execution.
- You have to parse the command line arguments (all infiles and any command line values) i.e. no hardcoding the names of the infiles or the value for interpolation task. This is because we will be using our own infiles, with different filenames and different locations.
- Plan before you write your code. Cover all possibilities regarding different inputs. Dynamically allocating structures for the infile contents is a must so that for infiles with more or less entries don't end up giving you errors during submission.

7.3 Running on Dimefox

You must first transfer your files from your home computer to the Dimefox server using the `scp` protocol. Then, log into dimefox and transfer the files to a relevant folder: perform the following set of commands on the terminal from your home location on dimefox (making the right folders and transfering the files in the right location):

```
mkdir ENGR30003
cd ENGR30003
mkdir A2
cd A2
cp ../../*.c .
cp ../../*.h .
cp ../../*.csv .
cp ../../*.pdf .
```


Remember to check this folder contains only the .c or .h files (if you use multiple c files and h files) you need for the assignment and the PDF report you wish to submit. Then try compiling your code using `gcc` on the terminal from the `A2` folder, to see if it works. The following compilation procedure must return no errors or warnings:

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

You can test your compiled code then using:

```
./exec in_shock.csv in_linalsys.csv in_interp.csv 4 in_waveeqn.csv
```

There are 5 command line arguments here, one for each of the 4 coding functions. The argument number 4 (4) is part of the interpolation task: the value of x_o at which the interpolated value is to be outputted. Once your code works for the provided infiles, it would help you if you changed the infiles and the 4^{th} argument to different values and see if the code still works and outputs acceptable results. If this works, your code is now submission worthy.

7.4 Submitting on Dimefox

All submissions must be made through the `submit` program. Assignments submitted through any other method will not be marked. Transfer your files to your home drive on dimefox. Check the transfer was successful by logging into dimefox and doing the `ls` command on the terminal. Once you've tested the code by running on Dimefox using the approach described in Running on Dimefox section, you can then submit your files using the command `submit` as follows:

```
submit ENGR30003 A2 *.c *.h *.pdf
```

Do **NOT** submit your infiles as this would likely corrupt your submission and would take time to fix. Wait for a few minutes and then carry out the verify and check steps:

```
verify ENGR30003 A2 > feedback.txt
nano feedback.txt
```

Look through this text file to check (a) your program compiled (b) it executed without error. Please read `man submit` on `dimefox` for confirmation about how to use `submit`, especially how to `verify` your submission. No special consideration will be given to any student who has not used `submit` properly.

You must also check that you have no memory leaks in your code as loss of memory from your implementation will result in deductions. You can use `valgrind` for this as follows:

```
valgrind ./exec in_shock.csv in_linalsys.csv in_interp.csv 4 in_waveeqn.csv
```

It must be pointed out that `valgrind` will take longer time to run compared with the normal execution, so, plan your submission accordingly. Incase your submission fails to pass the memory check, you will lose marks. There are two potential areas where you can lose marks: runtime error messages and heap/leak summary. Examples of runtime error messages include:

a. `Use of uninitialised value of size X`: Happens when you use a variable that has not

been defined or does not exist anymore or initialised.

b. `Conditional jump or move depends on uninitialised value(s)`: Happens when using an uninitialised variable to perform operations

c. `Invalid read/write of size X`: Happens when trying to scan or write to file a variable to memory which you do not have access to.

d. `Process terminating with default action on signal 11`: Happens when the error is so severe, your code is terminated automatically.

Examples of heap/leak summary errors are:

a.
```
total heap usage:  2,686 allocs, 29 frees, 152,138,664 bytes allocated
```

b.
```
==6504== LEAK SUMMARY:
==6504== definitely lost:  0 bytes in 0 blocks
==6504== indirectly lost:  0 bytes in 0 blocks
==6504== possibly lost:  0 bytes in 0 blocks
==6504== still reachable:  16,912,796 bytes in 2,657 blocks
==6504== suppressed:  0 bytes in 0 blocks
```

All submissions should be in C99, and use no functions outside of the C standard library and maths library. Some key points to consider about your submission and verification are outlined here:

- Submissions that can not be compiled or run by `submit` system 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. It would help if you don't write any additional files through your code. If it is absolutely necessary, then make sure the files do not exceed 2 MB individually. This would give errors during your submission.
- Since each task can be assessed individually, you can work submit your code with just one or two tasks implemented. The feedback will skip over tasks not implemented and only look at the outfile of the tasks implemented.
- Only your last submission is stored in the system i.e. everytime you use `submit`, 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.

8 Getting Help

There are several ways for you to seek help with this assignment. First, check the **Assignment 2 Frequently Asked Questions** wiki in the LMS (subsection Assignments). It is likely that your question has been answered here already. You may also discuss the assignment on the "Assignment 2" discussion board. However, please do **not** post any source code on the discussion board. You may also ask questions during the workshops or send me (Professor Sandberg, `richard.sandberg@unimelb.edu.au`) an email directly.

Note: Students seeking extensions for medical or other "outside my control" reasons should email Professor Sandberg, `richard.sandberg@unimelb.edu.au` as soon as possible after those circumstances arise.

