ENGR30003: Numerical Programming for Engineers
Assignment 1

August 25, 2019

Marks: This assignment is worth **25%** of the overall assessment for this course.
Due Date : Friday, 13 September 2019, 5:00pm, via `submit` on `dimefox`. 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.

Learning Outcomes

This project requires you to demonstrate your understanding of dynamic memory, linked lists and basic numerical computation. The key objective of this assignment is to solve a set of tasks which involve processing of flow around a flat plate.

Flow Around a Flat Plate

In the field of Fluid Mechanics, flow around a flat plate perpendicular to the flow direction is still an active area of research. With advent of high performance computing and supercomputers, it has now become possible to look at this simple case with a greater deal of accuracy. The problem consists of a flat plate that is perpendicular to the main flow direction as shown in Figure 1 (left). The blue arrows indicate the direction of the flow while the shaded object is the flat plate. This generates a wake behind the flat plate and exerts a pressure force on the plate, similar to the force you feel when you hold your hand out in a moving car. At a given instant the flow behind the flat plate is extremely complex and a snapshot of the flow domain is shown in Figure 1 (right).

Working with the Data

For this assignment, you will process the wake data from a flat plate case. The data has been provided to you in a CSV format file (`flow_data.csv`) with the following form:

```
rho,u,v,x,y
0.951,1.05,0,-15,-20
0.951,1.05,0.00155,-14.6,-20
0.951,1.05,0.00273,-14.2,-20
0.95,1.05,0.00366,-13.8,-20
0.95,1.05,0.00434,-13.4,-20
0.95,1.05,0.00467,-13.1,-20
0.95,1.05,0.00462,-12.7,-20
```

Each line corresponds to a point in the flow domain with coordinates `(x,y)`. The density `rho` and velocities at that given point in `x` and `y` are given by `u` and `v` respectively.

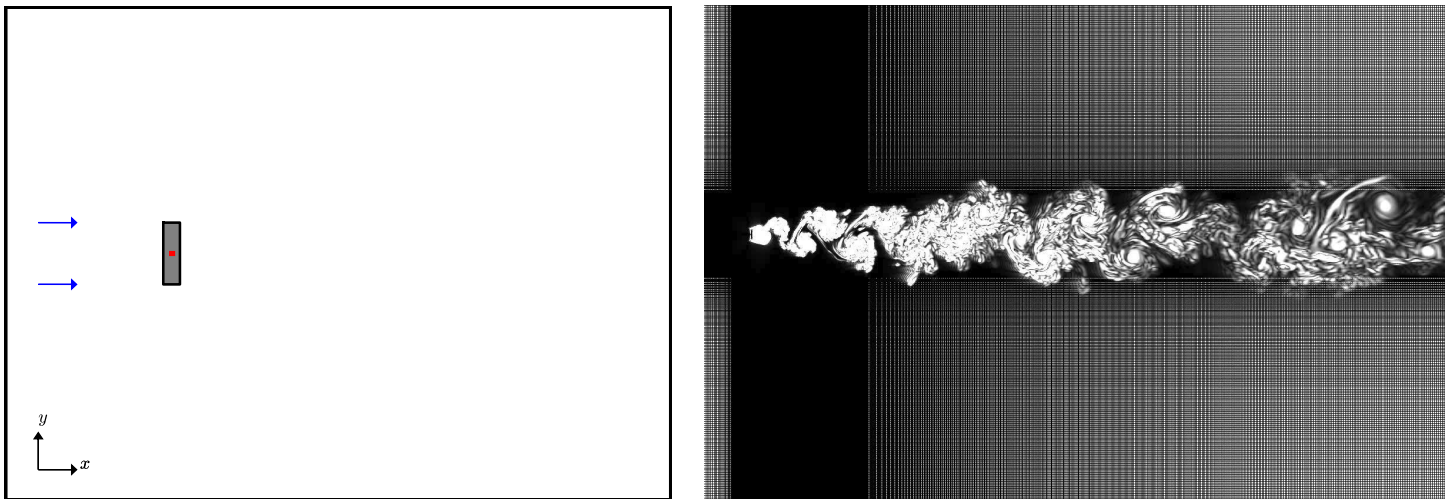

Figure 1: (left) Schematic of problem domain (right) Instantaneous flow field behind a flat plate superposed on the grid

Positive value of `u` indicates velocity direction is along the `x` direction while negative `u` indicates velocity direction is along `-x`. Similarly for the `v` velocity sign.

Processing Tasks

This assignment consists of four processing tasks which will be assessed independently. For each task you are to measure the run time it takes to complete the described task using your program (see program output below). **Each of the four tasks must not require more than 60 seconds** to run on `dimefox` using `valgrind`. This means, in order to complete the task within this time limit, you may need to focus on the efficiency of your solution for each problem. Overall you have to write a single program, performing each of the four tasks sequentially. For each task you have to write your results to a file on the disk. Details on using `valgrind` is available in the "Implementation" section of this assignment.

Task 1: Maximum Flux Difference [2/25 Marks]

It is sometimes helpful to understand what's the range of velocities in the flow. For the first task, you must compute the maximum flux difference, i.e., `rho*u` and `rho*v` after coordinate `x = 20`. Specifically, you must output first the two points in the domain where the magnitude of the `rho*u` flux difference is maximum followed by the two points in the grid where the magnitude of the `rho*v` flux difference is maximum. For each set of points, the point with the maximum of the given flux must be first followed by the point with the minimum flux. The output should be to the file called `task1.csv` and should be formatted as below. There must be no blank spaces between the values and around the commas. Each value must be written to 6 decimal places.

It is imperative that you write it out the way described above and shown below otherwise comparing your output to the solution would result in an error and you would lose marks.

```
rho,u,v,x,y
1.2438621,1.007986,-0.001002,40.512346,-19.595387
1.0148121,0.850117,0.0005807,66.899192,-0.729056
1.1438621,0.852483,0.0004275,69.552467,-0.729056
1.1386456,0.838355,-0.0006330,60.961891,0.442134
```

The above is an example of what the file should look like and is not the actual solution. Also note that the data provided in `flow_data.csv` is not in any chronological order and you must efficiently look only at points where the value of `x` is greater than 20. You can use `file_io.c` to understand how to output data to a file.

Task 2: Mean Velocities on a Coarser Grid [5/25 Marks]

Each line in the file `flow_data.csv` is a point location in the domain. These points when joined together will create a mesh (also called a grid). For this task, you will map these points onto a coarser grid, computing the new average coordinates `(x,y)` and the corresponding mean density `rho` and velocities `(u,v)`. The flow domain can be thought as divided into a two-dimensional grid such that each cell of the grid would contain multiple points, the number of which would depend on the cell upper and lower dimensions and the coordinates of the points. You would compute the average coordinates, density and velocities for each cell using the formula below for all points k within a given cell (this is for the `x` coordinate; same formula to be used for `y, u, v, rho`):

$$x_{av} = \frac{\Sigma_{i=0}^{k-1} x_i}{k}$$

While computing the averages, you must ignore any data points that lie on the coarse cell boundaries, i.e., only consider contribution of points that are wholly contained within the given cell. The resulting averaged values of the given cell can be obtained from a score S, computed by

$$S = 100 \frac{\sqrt{u_{av}^2 + v_{av}^2}}{\sqrt{x_{av}^2 + y_{av}^2}}$$

Finally, you must output the results of the averaged values and the score to `task2.csv` in descending order based on the score for each cell. An example of what the output should look like is shown below. There must be no blank spaces between the values and around the commas. Any float value must be written to 6 decimal places as shown:

```
rho,u,v,x,y,S
1.191265,0.831445,0.019688,69.842187,5.861905,1.186624
1.236148,0.874429,-0.001291,79.552381,9.923571,1.090734
1.234881,0.868093,-0.003071,79.552381,5.861905,1.088278
1.236199,0.861106,-0.001160,79.552381,-9.886786,1.074177
1.236118,0.864510,0.000087,79.552381,14.017391,1.070231
```

The size of the grid (number of cells in each direction) must be an input parameter, allowing the code to run different grid sizes. Your implementation would be checked for the grid resolution of 10 i.e. 10 cells in `x` and 10 cells in `y`. The domain extent for this coarse grid in `x` and `y` is -15 to 85 units and -20 to 20 units respectively. An example of the coarse grid is shown in Figure 2 (left). The 10 cells in `x` direction would span from -15 to 85 units while the 10 cells in `y` direction would span -20 to 20 units. Also shown is an example of a cell within this grid. As can be seen, the cell is of width Δ`x` and height Δ`y` and the black dots show the points in the original grid. Once you do the averaging for all these points, you will end up with one average point (shown in red).

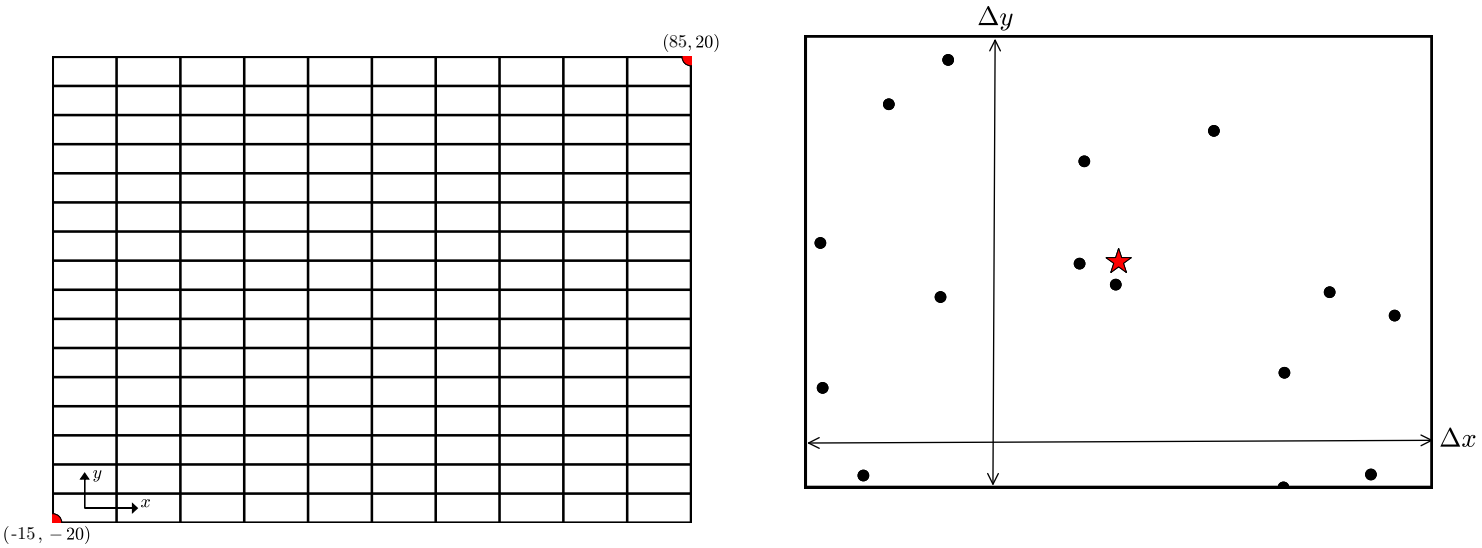

Figure 2: (left) Example of coarse grid with minimum and maximum extent shown by the coordinates (right) Example of a given cell where the black dots show the original points and the star shows the location of the averaged point

Task 3: Searching in Data Structures [8/25 Marks]

For this task, you will implement three data structures: an array, a linked list and a perfectly balanced binary search tree. The data contained in these data structures will be the same, with the aim to perform a search operation and output the time taken to search for each case. The data required for this task is a subset of the data in `flow_data.csv`. First, you must pull out the data points at the domain centreline, i.e. points where `y` = 0.0. You should store these points in an array. Next, you must sort the data points in ascending order of the streamwise flux `rho*u`. Then, you must insert the sorted values into a linked list and in a perfectly balanced BST. Finally, you must search for the data point in all the data structures whose `rho*u` value is closest to 40% of the maximum `rho*u` flux. You must first find out the exact value which is closest to the 40% of maximum `rho*u` in the array and then search for this exact value in the remaining data structures. The following outputs are required in the file `task3.csv`:

1. Linear search on the array to find the value. You must output all `rho*u` values upto and including the value that is being searched for.

2. Binary search on the array to find the value. You must output all `rho*u` values upto and including the value that is being searched for.

3. Linear search on the Linked List to find the value. You must output all `rho*u` values upto and including the value that is being searched for.

4. Search on the balanced BST for the value. You must output all `rho*u` values upto and including the value that is being searched for.

A sample output for the `task3.csv` file is shown below, where each line (total of 4 lines) corresponds to the cases in the order shown above:

```
3.614885,3.564577,3.562721,3.544763,3.542255,3.489632,3.488729
```

```
3.614885,3.564577,3.544763,3.542255,3.488729
3.614885,3.564577,3.562721,3.544763,3.542255,3.489632,3.488729
3.614885,3.564577,3.544763,3.542255,3.488729
```

The values above do not correspond to any data values but are provided only as a reference. The search time for all four cases must be written to standard output, in the format described in the Program output section. Here, perfectly balanced refers to nearly perfectly balanced i.e. there may be uneven leaf nodes but the length of these nodes must not be greater than other nodes by more than 1 depth. The search algorithm in all cases must terminate when the exact value is identified and the last value written out must be this value.

Task 4: Computing the vorticity [5/25 Marks]

One of the quantities of interest for engineers to study is vorticity, which represents the rotation of the velocities about an axis. The vorticity for the given data can be defined as follows:

$$\omega = \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right) \tag{1}$$

To compute the vorticity, there are several steps you have to follow. Since each line of the total $n * m$ lines of the data in `flow_data.csv` represents a point in the domain, you would first have to arrange these datapoints into a grid representation such that accessing any point in the domain is done through two indices, similar to a 2D array representation. It is important that the arrangement of the datapoints is consistent with the domain. So, if as an example, the `u` datapoints were put into a 2D array `U` of shape $n \times m$, then, increasing `i` in `U[i][j]` for a given `j` represents increasing `x` values and increasing `j` in `U[i][j]` for a given `i` represents increasing `y` values. Once, this is done, you can then compute the vorticity using the following formula:

$$\omega[i][j] = \left(\frac{V[i+1][j] - V[i][j]}{X[i+1][j] - X[i][j]} - \frac{U[i][j+1] - U[i][j]}{Y[i][j+1] - Y[i][j]}\right) \tag{2}$$

Here, ω, `U`, `V`, `X` and `Y` are defined as 2D arrays containing values of ω, `u`, `v`, `x` and `y` in the 2D domain. The indices in the above formula for `i` go from $0 : n-2$ and for `j` go from $0 : m-2$. For datapoints with indices $n-1$ or $m-1$, the value of vorticity will be given by:

$$\omega[n-1][j] = \left(\frac{V[n-1][j] - V[n-2][j]}{X[n-1][j] - X[n-2][j]} - \frac{U[n-1][j+1] - U[n-1][j]}{Y[n-1][j+1] - Y[n-1][j]}\right)$$

$$\omega[i][m-1] = \left(\frac{V[i+1][m-1] - V[i][m-1]}{X[i+1][m-1] - X[i][m-1]} - \frac{U[i][m-1] - U[i][m-2]}{Y[i][m-1] - Y[i][m-2]}\right)$$

$$\omega[n-1][m-1] = \left(\frac{V[n-1][m-1] - V[n-2][m-1]}{X[n-1][m-1] - X[n-2][m-1]} - \frac{U[n-1][m-1] - U[n-1][m-2]}{Y[n-1][m-1] - Y[n-1][m-2]}\right)$$

Once you've obtained the values of ω, you must report the number of datapoints that have the absolute ω values within a certain threshold. Thus, you must report the number of datapoints in the following format to the file `task4.csv`:

```
threshold,points
5,10000
10,20000
15,50000
20,5000
25,2500
```

The number of points shown are only for reference and do not refer to the solution. The thresholds here mean 10,000 points have absolute ω between 0 and 5, 20,000 points have ω between 5 and 10 and so on. You may assume the data in `flow_data.csv` is sorted in ascending order in `y` followed by ascending order in `x`.

Implementation [5/25]

The implementation of your solution is extremely important and your implementation should be based on the provided skeleton code. A total of [3/25] marks are allocated to the quality of your code:

- Program compiles without warning using `gcc -Wall -std=c99` on dimefox
- Exception handling (check return values of functions such as malloc or fscanf)
- No magic numbers i.e. hard coded numbers in your code (use #define instead)
- Comments and authorship for each file you submit are important (top of the file)
- General code quality (use code formatters, check for memory leaks)
- No global variables

There is a coding etiquette we are enforcing for this assignment, which is concerned with the comments in the code. Every code snippet must be provided with sufficient comments, i.e., what does the following block of code do. This is imperative in case your output is incorrect; as it makes it easier for the grader to assess if your implementation was correct. The quality of comments are allocated a total of [2/25] marks. The following sections describe the required command line options and program output of your program.

Command-Line Options

When running on `dimefox` your program should support the following command-line options:

```
terminal:  gcc -Wall -std=c99 *.c -o flow -lm
```

for the compilation and

```
terminal:  valgrind ./flow flow_data.csv 10
```

for the execution, where `flow_data.csv` is containing the flow related data and 10 is the grid resolution.

Program Output

When running on `dimefox` the program should only output the following information to stdout in addition to the csv files for each task:

```
terminal:  valgrind ./flow flow_data.csv 10
TASK 1:  200.63 milliseconds
TASK 2:  14063.82 milliseconds
TASK 3 Array Linear Search:  30.4 microseconds
TASK 3 Array Binary Search:  40.5 microseconds
TASK 3 List Linear Search:  100.21 microseconds
TASK 3 BST Search:  50.1 microseconds
TASK 3:  209.46 milliseconds
TASK 4:  221.16 milliseconds
```

Note the units of time in the output for the tasks. For each task, output the time, in relevant units, taken to perform all computation associated with each task. This can be achieved by adopting the `gettimeofday.c` code available in the resource for Week 2. After the program is executed, the files `task1.csv` , `task2.csv`, `task3.csv` and `task4.csv` containing the results for the different tasks should be located in the current directory.

A total of [1/15] marks is allocated for correct implementation of the output format in terms of console output and output written to the result file generated by each task.

Provided Code

The following files are provided to you for this assignment:

- `main.c`, where the parsing of data from command line is to be done and timing for each task implemented.
- `tasks.c`, where you would implement four functions `maxfluxdiff()`, `coarsegrid()`, `searching()` and `vortcalc()`, for each task.
- `tasks.h`, which need not be changed and acts as a header file to link the C file to the main file.

You are free to use guide programs provided during the lectures on the LMS and adapt them for your use. This could be any of the files like `file_io.c`, `linkedlist.c`, `bst.c` and more. You may also use the header files if needed. Remember to fill in your details in each file you write, such as the student name and student number.

Points to consider

- Only use type `int` and `double` for integers and floating point numbers respectively. When writing out a float to output make sure the format restricted to a 6 decimal place number. Writing to file needs to be done according to the format given in the assignment to be marked correct by the system. If there are multiple numbers on a line separated by a comma, leave no space in between.

- Each task is independent of the other so you can work on each task individually and test it out. Make sure you adhere to the formatting and output file headers as shown in the Assignment.
- Read the data into an appropriate data structure for each task. You may choose to read it in once and reuse this structure for all tasks but the task functions might need to be modified.
- Write the header for each file as described in the assignment. Your solution would be marked wrong by the system even if you have the right solution if you get your headers wrong.
- Your code should not contain any magic numbers. Examples include but are not limited to using `malloc(1000 * sizeof(int))` or `for (i = 0 ; i < 20; i++)`. Use `#define` at the top of the file to define 1000 and 20.
- Remember to free your data structure and any other structure you allocate dynamically. Statically allocated structures are not allowed.

Submission

You need to submit your program for assessment. Submissions will **not** be done via the LMS; instead you will need to log in to the server `dimefox` and submit your files using the command `submit` . You can (and should) use `submit` **both early and often** to get used to the way it works, and also to check that your program compiles correctly on our test system, which has some different characteristics to the lab machines. Only the last submission will be marked. The submission server may be very slow towards the deadline as many students are submitting. Therefore, please do not wait until the last few minutes to make the first attempt of submission. If you make a submission attempt a few minutes before the deadline but the submission was completed after the deadline, your submission will be treated as a submission AFTER the deadline.

All submissions must be made through the `submit` program. Assignments submitted through any other method will not be marked. Transfer your code files to the home drive on dimefox. Check the transfer was successful by logging into dimefox and using the `ls` command on the terminal. 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 A1
cd A1
cp ../../*.c .
cp ../../*.h .
cp ../../flow_data.csv .
```

Here you're making the `A1` folder within the `ENGR30003` folder and then moving to the `A1` folder. From this folder, you move the files transferred from the home to the `A1` folder. Remember to check the `A1` folder contains only the .c or .h files (if you use multiple c

files and h files) you need for the assignment and the CSV data file. Then try compiling your code and executing it to see if it works without errors. Once you're satisfied with your files, you can submit the files (only the c and h files, not the csv) via the `submit` system on `dimefox` as follows:

```
submit ENGR30003 A1 *.c *.h
```

Wait for a few minutes and then carry out the verify and check steps:

```
verify ENGR30003 A1 > feedback.txt
nano feedback.txt
```

Look through this feedback text file to check (a) your program compiled (b) it executed without error. Please read `man -s1 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. Your submissions will be assessed via `valgrind` on `submit`. To use `valgrind` to check your implementation, you must execute the compiled file using:

```
valgrind ./flow flow_data.csv 10
```

Since `valgrind` is a memory error detector, it will be used to assess your submissions for potential memory misuse. A well-implemented code will have the following output:

```
==3887== HEAP SUMMARY:
==3887== in use at exit:  0 bytes in 0 blocks
==3887== total heap usage:  53 allocs, 53 frees, 56,921,168 bytes allocated
==3887== All heap blocks were freed -- no leaks are possible
```

It must be pointed out we are not using a small data file so `valgrind` will take longer time to run compared with the normal execution. 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:

1. `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.
2. `Conditional jump or move depends on uninitialised value(s)`: Happens when using an uninitialised variable to perform operations
3. `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.
4. `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:

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

2. ```
   ==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
   ```

You may discuss your work during your workshop, and with others in the class, but what gets typed into your program must be **individual** work, not copied from anyone else. So, do not give hard copy 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 program, 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 are so identified will be referred to the Student Center.* See `https://academichonesty.unimelb.edu.au for more information`.

Getting Help

There are several ways for you to seek help with this assignment. First, go through the **Assignment 1 Important Notes** in the LMS. It is likely that your question has been answered there already. Second, you may also discuss the assignment on the **Assignment 1 Questions** discussion board. However, please **do not** post any source code on the discussion board. Finally, you may also ask questions to your tutors during the workshops and if it is still unresolved, contact either the lecturer Thomas Christy (`thomas.christy@unimelb.edu.au`) or the head tutor Chitrarth Lav (`chitrarth.lav@unimelb.edu.au`) directly.
   ```