1 Introduction

Lagrangian and Eulerian models are most widely used to simulate advection-diffusion equations with computational particles. Where Eulerian models refer to a system that is fixed, Lagrangian models refer to a system undergoing instantaneous fluid velocity. The motivation for this project is to model the advection-diffusion of milk in coffee by coupling the Lagrangian method with the Eulerian numerical approach. This report summarises the following; the method used to simulate this model, the tests ran to validate the method used, and lastly states any improvements that could be made. For accessibility, a user interface code was written which allows the change of certain parameters to address different engineering questions.

The Lagrangian particle-based method produces a stochastic model that represents each particle following a trajectory in a fluid undergoing Brownian motion in a closed boundary over time. This is modelled through the use of stochastic differential equations with the Wiener process. Gaussian distribution and intrinsic python modules are used to ensure the particles have random initial locations.

Using the Euler-Maruyama approach, the particle's new position can be computed based on a homogeneous spatial distribution of turbulent velocity fluctuations which specifies the distance that the given particle has travelled based on the time-step. This simulation is presented as a 2D Eulerian concentration field (ϕ) to give a complete view of the motion of particles over time.

2 User Interface

The User Interface (UI) is designed to simplify the interaction between the person and the simulation program as well as make it more enjoyable. This is done by giving the user the ability to change parameters in a separate window which opens when the code is run, as well as see the results visually in real time by making the program plot every time-step.

A 'tkinter' import was used to produce an interface that is easily accessible to anyone. The interface will ask the user to input the details they want to run the simulation on. The window is self explanatory, even for a first time user. Although 3 files were provided, the simulation can be started by running the 'main.py' file alone (as long as all 3 files with the appropriate velocity and reference data are within the same folder). This reduces the chance of the user accidentally changing something crucial in the code.

3 Validation

The validation task requires a one dimensional (1D) simulation, with zero velocity, diffusivity = 0.1, and a grid $N_x, N_y = 64, 1$ that ends at t = 0.2s, starting from initial step conditions:

$$\phi(x, y) = 1, \quad x \leq 0$$
$$\phi(x, y) = 0, \quad x > 0$$

Using the user interface, the parameters of this simulation are altered to the above conditions. Random points are initialised, and a 'while loop' is triggered that contains 'counter(), plot(), density_phi()' functions to do the following, respectively: calculate new positions of particles, plot those particles, and lastly plot the graph of density vs x-position and compare it with the reference file (reference.DAT). This is used to produce Figures 1 and 2.

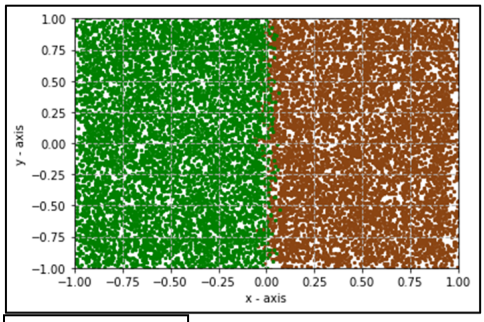

Figure 1

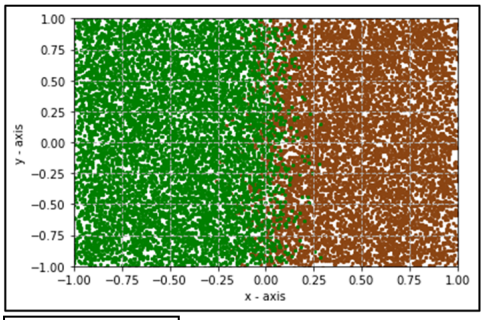

Figure 2

Figures 1 and 2 are the Lagrangian graphical representations of the particles in a 1D diffusion between the green particles (milk) and brown particles (coffee). Figure 1 shows the initial conditions stated whereas Figure 2 is at t = 0.2s and shows the particles slowly beginning to diffuse into each region. The running time of the simulation was recorded using the 'timeit' python package.

The reference data is provided in discrete points and must be interpolated to determine the unknown values at a given point. This was done using the python module 'interp1d' which interpolates the 1D arrays, 'ref_solution_x' and 'ref_solution_y' and returns a function f(x). This function can be given any x value within the range of the reference.DAT file and will return the value of y as a result.

The data is plotted to show the concentration of milk against the position on the x-axis by calculating the concentration of the green particles on every division of the 64 x 1 grid separately. Figure 3 shows a comparison of the experimental data and the reference solution provided to validate the test. It is shown that the experimental data fits closely to the reference solution, and therefore the experimental data is validated.

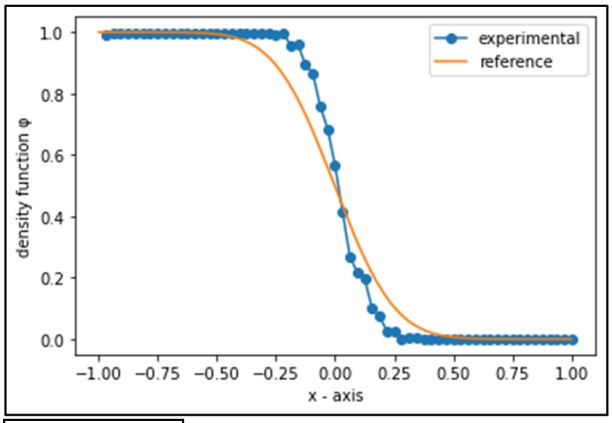

Figure 3

The global error is evaluated by computing the Root Mean Square Error (RMSE) between the numerical (simulation) solution and the reference solution. The simulation program was used to determine the values of RMSE for increasing number of particles as well as for increasing time-step. These values were then transferred to excel and plotted. The following graphs are shown.

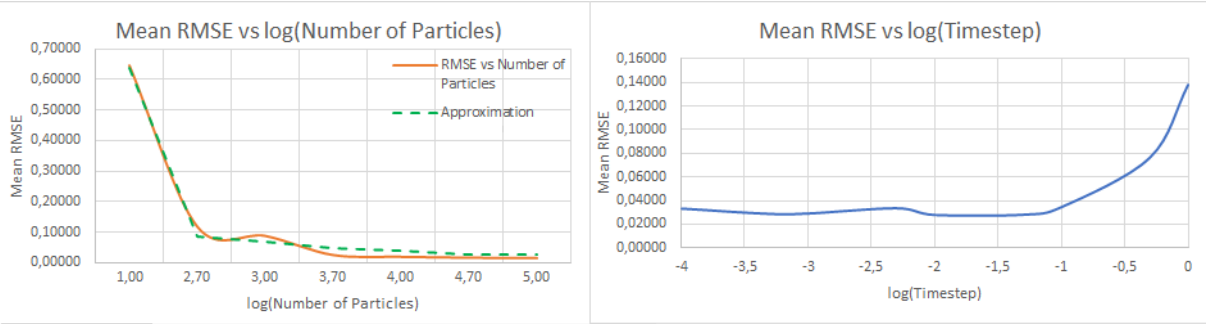

Figure 4

For every RMSE value, the program was ran 3 times and then the average was taken. In order to make the graphs easier to understand, the x-axis was done in a log-scale. For the graph on the left in Figure 4, the simulation was running until t = 0,2s with a constant time-step h = 0,005s, while changing N_p. For the graph on the right in Figure 4, the program was also running until the simulation time took 0,2s, however this time, the number of particles N_p was kept constant and equal to 5000, while h was varying. One can notice that for increasing number of particles, the RMSE clearly decreases while for increasing time-step it clearly increases. For the graph of RMSE vs log(No. Particles) an approximate curve fitting function was used of the form E=aN^{β}, where 'a' and 'β' are constants that needed to be found. For this Simulation program, the approximate curve-fitting function for the RMSE error based on N is thus:

$$f(N) = 0,63517 * N^{-2}$$

Where N is the log of the number of particles N_p.

4 Engineering Simulation

The main aim of this task is to locate the specific area where the value of concentration is more than 0.3 at a longer period of time after the oil spill happened. The velocity field file, velocityCMM3.DAT is also implemented in the code to act as a sea current that keeps the particles moving constantly. The initial state of the patch of chemical is stated below:

$$\begin{aligned} Concentration, \quad & \phi = 1 \\ Diffusivity, \quad & D = 0.1 \\ Positions, \quad & x, y = 0.4, 0.4 \\ Radius, \quad & R = 0.1 \\ Gridsizes, \quad & Nx, Ny = 64, 64 \\ No.Particles, \quad & Np = 150000 \end{aligned}$$

All the initial values can be inserted into the UI provided in the user_inputs.py file. Below is a list of the most important functions for the oil spill task with descriptions of what they do.

random_initial_values() - This function generates the initial random positions of two different colored particles. In the case of the Engineering Simulation, this function will append green particles (oil) into a specific circular boundary of radius R = 0.1 at the position x,y = 0.4,0.4 within the graph while everything else is filled with brown particles (water).

velocity_profile_separation_into_arrays() - This function opens the velocityCMM3.DAT file and divides the variables into four arrays: x_database, y_database, u_database and v_database. This later allows for interpolation into a 2D velocity field from which one can extract the velocities in the x,y direction, given the x,y position within the graph. In the case of the Engineering Simulation, since the velocity field needs to be turned on, this function is of upmost importance.

Euler_Maruyama_x_position(), Euler_Maruyama_y_position() - These two functions calculate the new x,y positions respectively. Depending on the user input, these functions can either introduce the velocity field function into the calculation of the new position or not. Additionally, the functions contain wall boundaries for the particles to not exceed the graphing boundaries and keep N_p constant. For the case of the Engineering Simulation, these two functions calculate the new x,y positions of particles including the velocity field function.

counter() - This function takes the x,y positions of both the water and oil particles as inputs and finds the new positions of those particles by calling the Euler_Maruyama functions. Then it appends those new values into new positional arrays x_new and y_new respectively. It then clears the original arrays and appends the new values into the original arrays. This manipulation results in very quick calculations as the past particle positions are never stored after they have been plotted for the user. For the Engineering Simulation this allows for the visualization of the movement of the oil spill.

density_phi() - This function produces two 2D histograms, one for the water particles and one for the oil particles. It then calculates the concentration of oil within the graph using the following equation:

$$Concentration, \quad \phi = \frac{colored\_particles}{particles + colored\_particles}$$

Where colored_particles and particles are 2D histograms for oil and water respectively. The function then produces a mesh-grid (x_grid,y_grid) from the 2D histograms and plots it with concentration using the pcolormesh() function.

timeit() - This function gives us the time that the simulation took to solve the problem. It is important to state that this time does not correlate with the simulation time but describes the real life time it took to execute the code.

Using the above functions and the initial variables, the simulation time (t) was set to 50 seconds using a time-step (h) of 0.1s and after a longer period of time, these were the results (Figure 5):

```
The coordinates of points where concentration > 0.3
[-0.15625, 0.28125]
[-0.09375, 0.34375]
[0.21875, 0.40625]
Time taken to run simulation:  3137.6648761999995
```

Figure 5

Thus, we can use inductive reasoning to state that no matter how long the simulation will run for, there will always be a patch where the concentration of oil is greater than 0,3.

5 Method Improvement

In order to improve the current state of the simulation and make it quicker, the following can be done:

- Increase the size of the particles and thus, decrease the number of particles. This will make the process quicker as there are less calculations done. It might not give a more correct numerical solution, however the visual aspect would be faster.

- Make the graph circular (like a cup of coffee!) which would allow the user to just focus on the most important region of the simulation. The area would be smaller as the circle would be inscribed into the current square graph and less particles would be needed.

- Plot only the final result (density field and not the particles separately) instead of every time-step.

- Plot only the separate (milk/oil) particles in interest at the time steps.