COMPUTATIONAL METHODS AND MODELLING ASSIGNMENT:

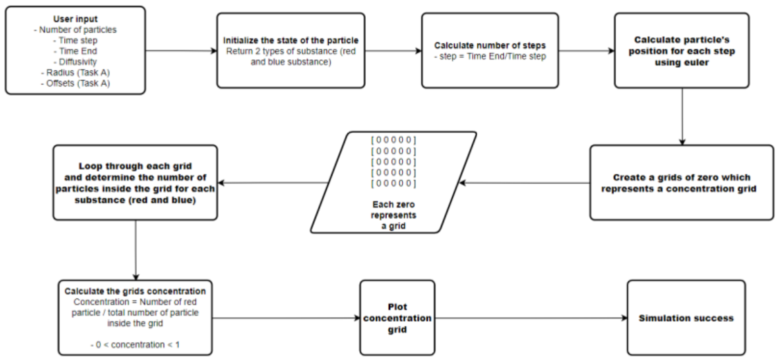
ADVECTION AND DIFFUSION OF SUBSTANCE

Group 4

Members	Percentage contribution (%)
Khalid Hersafril	16.186
Sara Castellano	15.886
Sid Sharma	15.586
Tobias Humphrey-Evans	15.486
Isambard Wakefield	12.286
Daniel Round	12.286
Zoe Lee	12.286

Simulation of Advection and Diffusion of a Substance

Introduction


User input
- Number of particles
- Time step
- Time End
- Diffusivity
- Radius (Task A)
- Offsets (Task A)
Initialize the state of the particle
Return 2 types of substance (red and blue substance)
Calculate number of steps
- step = Time End/Time step
Calculate particle's position for each step using euler
Create a grids of zero which represents a concentration grid
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
Each zero represents a grid
Loop through each grid and determine the number of particles inside the grid for each substance (red and blue)
Calculate the grids concentration
Concentration = Number of red particle / total number of particle inside the grid
- 0 < concentration < 1
Plot concentration grid
Simulation success

Generally, the simulation runs by the following algorithm. The code uses **numpy, scipy and matplotlib** extensively which allows the simulation to render plots in a short amount of time. The simulation is constructed from a few classes which are **Globals, InitialState, SimulationMaths and Concentration** where it allows the simulation to be more powerful and robust. Some general description of what each class does are:

Globals	Where users input their desired values
InitialState	Initialized the condition for each task
SimulationMaths	Handles all the Mathematics required in the simulation
Concentration	Handles anything related to concentration

Task C (User interface)

The user interface for this simulation is only a script where the user would have to run the **main.py** file in order to perform the simulation. However, in order for the user to change some variables, they would have to change it in the **global.py** file. For more clarity, user can visit the following github page : https://github.com/KhalidOwlWalid/particle_simulation

Task A variables	Description	Options
self.task	Sets the task	A
self.Np	Sets the number of particles for our simulation	Any **int** value ($10^3 - \inf$)
self.h	Sets the time step for our simulation	Any **float** value (0 < h < 1)
self.tEnd	Sets the time to end our simulation	Any **float** value (tEnd > 0)
self.grid_size	Sets the grid size for our concentration plot	Any **int** value
self.radius	Sets the radius of our circle	Any **float** value
self.offset_x, self.offset_y	Sets the offset of our circle from the center	Any **float** value within the domain
Self.xMin, self.xMax	Sets the boundary for our x-axis	Any **int** or **float** value
Self.yMin, self.yMax	Sets the boundary for our y-axis	Any **int** or **float** value
Self.D	Diffussivity coefficient	Any **int** or **float** value
Self.include_velocity	Set true to include velocity in the simulation and And false to neglect velocity field	**True, False**
Self.plot_2D_particle (OPTIONAL)	Set true to visualize the particle plot	**True, False**
Self.size (OPTIONAL)	Adjust the size of our scatter points (larger values for larger points)	Any **int** or **float** value

Task B variables	Description	Options
Self.task	Sets the task	B
Self.Np	Sets the number of particles for our simulation	Any **int** value ($10^3 - \inf$)
Self.h	Sets the time step for our simulation	Any **float** value (0 < h < 1)
Self.tEnd	Sets the time to end our simulation	Any **float** value (tEnd > 0)
Self.grid_size	Sets the grid size of Nx (By default, Ny = 1)	Any **int** value
Self.rmse_plot	Plots the RMSE against parameter (Np and h)	**True, False**

By default, task D is solved using the already specified parameters: **self.radius, self.offset_x, self.offset_y, self.D, self.include_velocity** have been locally set inside **task_d.py**. The user however, is allowed to change other variables such as **self.Np etc.**

Task B

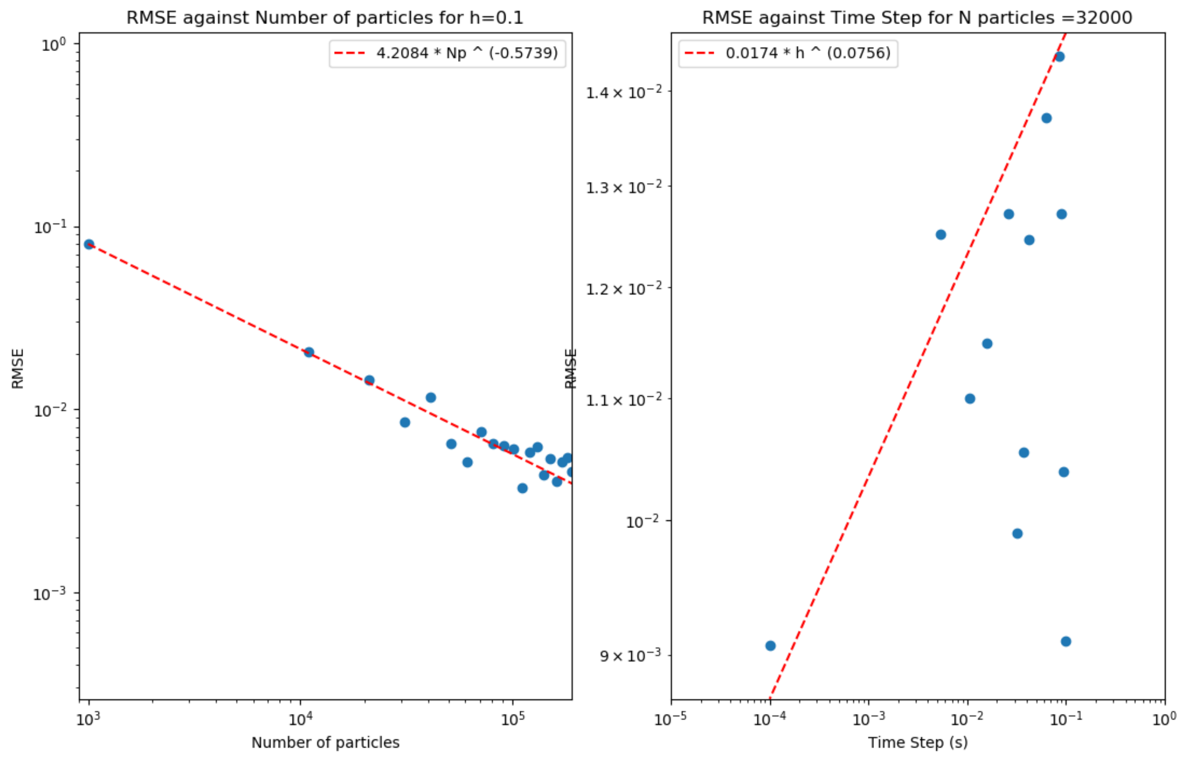


Figure 1 – RMSE plot for different parameters

The RMSE fitted equation for both parameters are generally in the form of $E = aX^B$

$$E_{N_p} = 4.2084 * N_p^{-0.5739}$$

$$E_h = 0.0174 * h^{0.0756}$$

$$\boldsymbol{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_{observed} - y_{actual})}{N}}$$

The **root mean square error (RMSE)** is calculated using the equation above, where $y_{observed}$ is the value obtained from the simulation, while $y_{reference}$ is an interpolated value derived from the reference solution, and N is the number of points used on our observed solution.

Since we are using stochastic approach, in order for us to approximate to a more accurate result, for each parameter (Np and h), and for each element inside the parameter (eg 1000,10000), we repeat the simulation 10 times for the same element and calculate the RMSE value for that element before finding the average RMSE value. This allows us to obtain a more reliable analysis. Since it is a stochastic approach, one can notice how random the distribution of the plot is from the **RMSE against Time Step** plot. Re-running the simulation multiple times will sometimes gives different values of RMSE and does not necessarily give the same average RMSE value. However, it would not affect much of our calculation since the average RMSE values for all elements in the time step analysis is small and can be neglected. You can refer **figure 3** to see the result of changing time step.

From the graph, it can be observed that as **the number of particles increases, the averaged RMSE value decreases** which means that the accuracy increases as we increase the number of particles whereas for the time step, average RMSE decreases as we decrease the time step. The factor of the RMSE value for different number of particles approximately decreases by a factor of 10 as we increase the number of particles by a factor of 10 specifically, from 1000 to 10000. As we increase it further, the average RMSE started to cluster. This is mainly because our functional form plateau as we increase the number of particles, hence any parameter changes made to the number of particles above 100000 would result in only a small increment of the average RMSE value. This relationship can be supported by the following diagram.

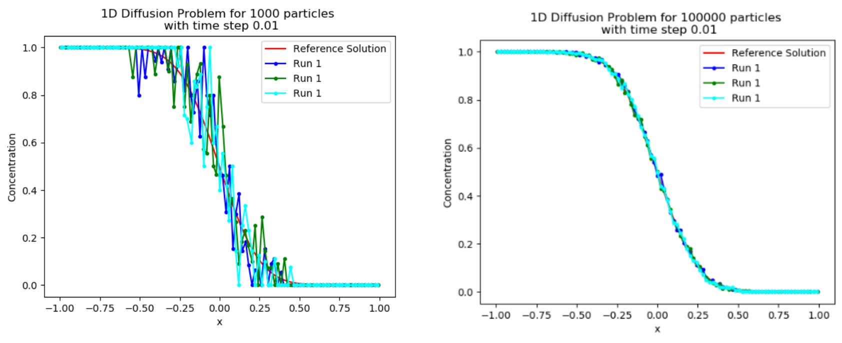


Figure 2 – 1D Concentration Plot

Here, we can see how increasing the number of particles could potentially allow us to estimate the concentration better with respect to the reference solution.

However, for different time steps, **decreasing the time step leads to a decrease in the average RMSE value**. One important thing to note that decreasing this parameter to a factor of 10 does not make any significant changes to our average RMSE value. It will only be a really small difference as shown in the graph. For instance, changing the parameter from 0.1 to 0.0001 would only lead to a difference of approximately $\mathbf{0.4\ x10^{-2}}$**.** To prove that this is indeed the case, observe **figure 3.** From rough observation, we can see that decreasing the time step even to a factor of 1000 (from 0.1 to 0.0001) does not give any significant changes to our plot. It can be said that changing the number of particles is a better option for us to get a more accurate result rather than changing the size of the time step. In terms of application wise, this is a great trade as we could possibly consider changing only the number of particles when trying to study the particle's motion which would reduce simulation time and cost.

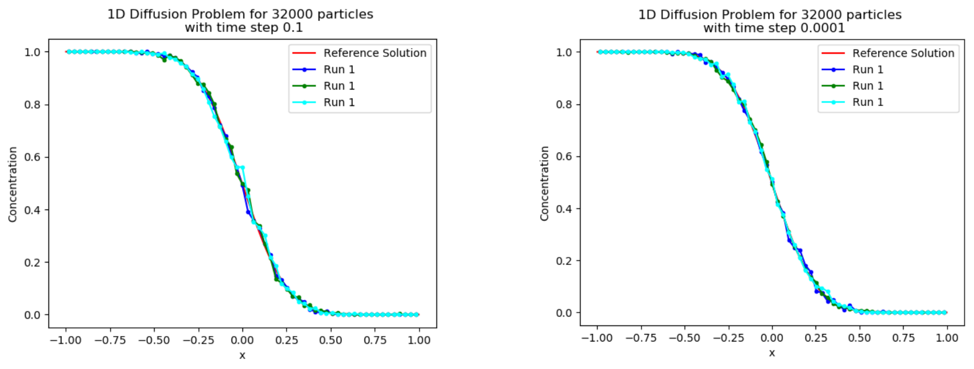


Figure 3 – 1D Concentration Plot for different time steps

Chemical Spillage Simulation (Task D)

Task D solves a specific engineering problem using the general approach in Task A. This section simulates the spillage of a chemical substance by interpolating the velocity to simulate the advection problem and by modelling the diffusion problem with the Eulerian approach. All plots shown are for 150 000 particles and 100 by 100 grids.

To illustrate the robustness of the code, we will discuss the algorithm implemented in the simulation. The initial state of the particles is inherited from the class **TaskA** where it initializes the particle's position to be either inside or outside a radius 0.1 with an offset of +0.4 and +0.4 from the centre in the x and y axis direction respectively. Particles within the radius and outside the radius are then assigned as substance 1 and substance 2 respectively.

The next position of the particle is determined by applying the particle based Lagrangian numerical method. This includes using the initialised coordinates, the velocity file, diffusivity coefficient and normally distributed random variables to simulate the Brownian motion of the fluid. We, then passed on arrays of the position of the initialized particle's coordinate to the Euler formula to obtain the next position of the particle and this calculation iterates through a number of steps to obtain the final coordinate of each particle's position until our desired time length. The algorithm then creates a grid of zeros with the dimension (Nx-1) and (Ny-1) where in this report, it has a dimension of 99 by 99 grids. From here, we will iterate through grid by grid and find the number of particles of each type of substance (substance 1 and substance 2) inside that grid and determine its concentration using the following equation:

$$Concentration = \frac{n(substance\ 1\ particle)}{n(substance\ 1\ particle) + n(substance\ 2\ particle)} \qquad [1]$$

where n(substance type particle) is the Number of particles of each type of substance inside a grid

The main finding is that the simulation models the advection-diffusion problem well, with errors associated with the selection of the time step. Overall, the simulation shows the movement of the particles in a circular shape (as expected from the nature of the given vector field) and random due to the random constant introduced to the equation. As we increase the time step, the numerical solution that are presented possess greater uncertainty as the local truncation error accumulates over time, hence introducing errors that are non-negligible which is called **global truncation error**. The global truncation error is proportional to the time step where it can be observed in **figure 4** where the simulation presents a spot in the centre where no particles are present, diverging from the actual functional value.

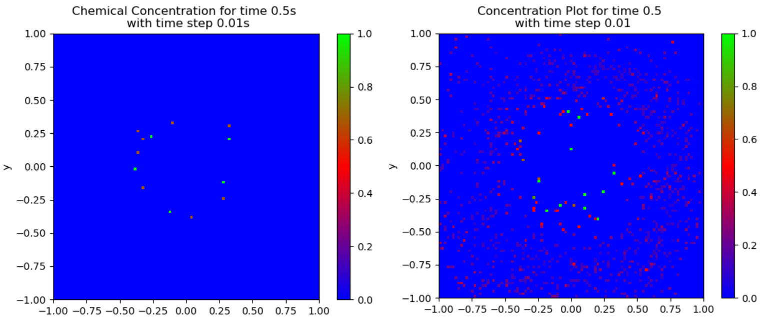


Figure 4 – Concentration Plot for time step 0.01 seconds

In this report, we will analyze two different time steps, which is 0.01 s and 0.001s. Performing the simulation with a time step of 0.01s, might be accurate for time less than 0.5s, but as the simulation keeps on running, it starts to deviate from the true behavior of the particle where the true behavior can be estimated with a time step of 0.001s.

From both figures, it can be observed that the concentration is stable after time = 0.5 seconds, as it completes one revolution. For a time step of 0.01 s, after running the simulation for a longer time, we will observe that the domain where the concentration is larger than 0.3 will stabilize around $-0.5 < x, y < 0.5$ whereas for a time step of 0.001s, the particles are well distributed throughout the whole domain, or to put it simply, the chemical concentration diffused and dissolved in the sea due to the current. However, this pattern starts to break as we run the simulation above time = 2.5s.

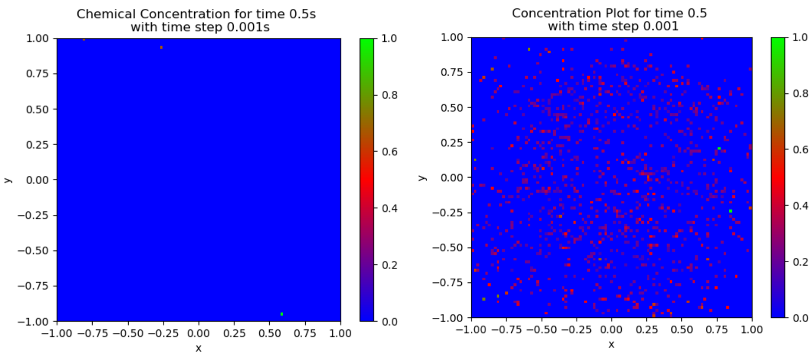


Figure 5 – Concentration Plot for time step 0.001 seconds

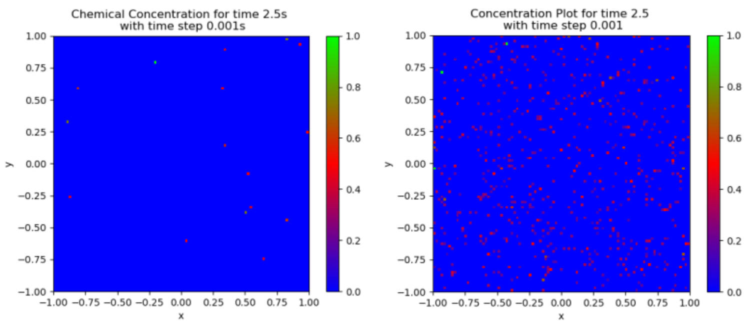


Figure 6 – Concentration Plot after 2.5 seconds for time step 0.001 s

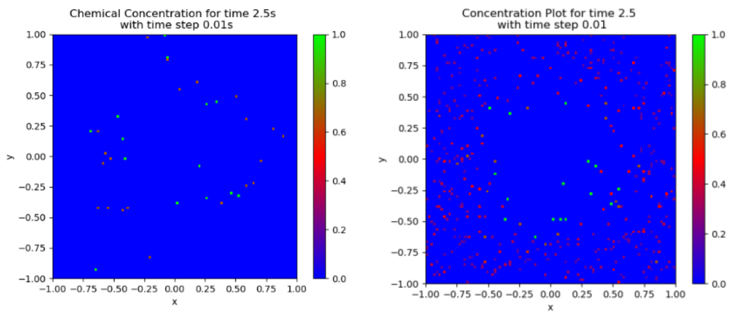


Figure 6 – Concentration Plot after 2.5 seconds for time step 0.01 s

Here, we can observed that as we increase the time of the simulation, due to our stochastic approach, the particles still move with the given vector field, but the randomness causes it to move in any direction which results in some of the particles to cluster together at some places, hence increasing the concentration to be above 0.3 at certain spots. This also applies to time step 0.01s, where over time, concentrations larger than 0.3 started to appear outside the $-0.5 < x, y < 0.5$ domain as we discussed earlier. One of the reasons for the concentration being greater than 0.3 consistently between the bounds previously mentioned, is that since the diffusivity coefficient is relatively small, the chemical spillage will diffuse quite slowly in the proximities of the centre of the domain, hence why for the first 2.5s, the pattern is the same. It is possible to see that if the diffusivity coefficient were to be bigger, the next particle's position would probably be further from the current particle's position, and as a result the chemical spillage would also move outside the $-0.5 < x, y < 0.5$ boundary at any time. Additionally, it is worth considering how the presence of random perturbations adds a deviation from the mean of the solution. In fact, it is possible to notice that for the same time lengths, different plots might be obtained where the concentration is greater than 0.3 outside of the expected domain.

Task E

As it is always of interest to improve computational speed, it would be beneficial to only consider one set of particles: that of the chemical spillage. If our main objective is to only study the particle's motion, it is not necessary to include all the particles during the simulation and only include the particle that we are interested to study. In fact, an engineering estimate may be made on the percentage of the total particles that are the chemical spillage substance, say 12% of the 150000. In this way, the computation will have to run only for $\mathbf{0.12 * 150000 = 18000\ \mathit{particles}}$ particles making the computational time faster. The current algorithm that we have implemented will run through each grid to check both type of particles (substance 1 and substance 2) which simply means that we would have to iterate through a total length of 150000 elements from arrays.

Proposal

The validation and engineering simulation will run in an analogous manner; however, the concentration calculation will be different. We would not be able to divide the number of particles of one substance by the total number of particles in that grid (refer equation [1]), since we do not have two sets of substances. Therefore, a different approach can be implemented. This consists of checking each grid if there is any particles of interest inside it. We will then calculate the concentration using the following formula:

$$Concentration = \frac{Number\ of\ particles\ of\ interest\ inside\ the\ grid}{Total\ number\ of\ particles\ of\ interest\ in\ the\ whole\ domain} \qquad [2]$$

Example:
Suppose we have 15000 particles of substance 1. Inside grid (30,30), we have a total of 2000 particles of the same substance. The concentration would then be:

$$Concentration = \frac{2000}{15000} = 0.133$$

Hence, the concentration of that grid would be 0.133.

Proof of concept

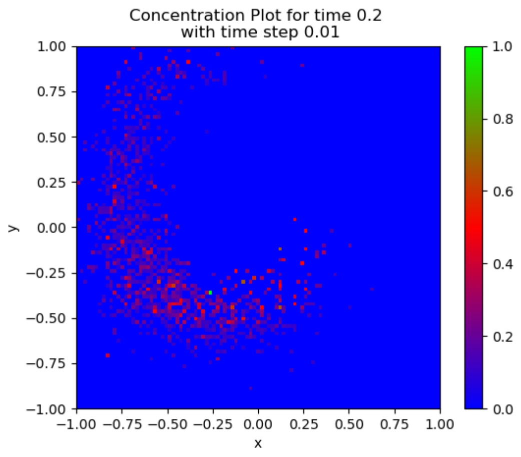


Figure 7 – Task A

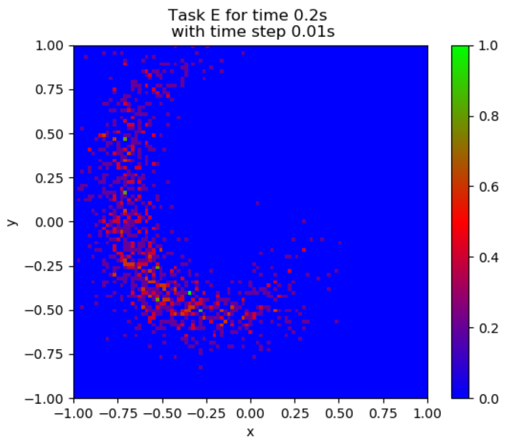


Figure 7 – Task E

The following is the input for this example:

Self.Np = 150000

Self.h = 0.01

Self.tEnd = 0.2

Self.D = 0.1

Self.radius = 0.1

Self.offset_x, self.offset_y = 0.4

The computational time to render the graphics for Task A is 19.72 meanwhile it only takes 3.2 seconds for Task E which is reduced by a factor of 6. The code is much more efficient since we are only considering only one substance unlike for task A where we also have to take into account the other substance to obtain its concentration plot.