Dynamic Brownian Simulation for particle transport in Python

School of Mathematical and Statistical Sciences
Summer Scholarship 2026

Máire Geraghty

Supervisor: Dr Niall Madden

1 Introduction

The aim of this project was to develop a Python-based *Dynamic Brownian Simulation* of particles in a two-dimensional channel with boundary absorption. Their motion is governed by a set of stochastic differential equations. It was inspired by a larger project being conducted by Niall Madden and Nanda Poddar; see, for example [3]. In that project, solutions to the advection-diffusion partial differential equation problem are computed using a Python-based FEM solver, Firedrake [1], and results are compared with solutions of a corresponding stochastic formulation. This stochastic model, i.e., the Brownian Dynamic Simulation, was developed in a different coding language called MATLAB [2].

However, comparing two models written in two different coding languages, and running on different platforms, is not the effective. On the one hand, MATLAB is an academic coding language specifically designed to make plotting data and analysing it easier, along with modelling complex mathematical problems. The disadvantage of MATLAB, however, is that the free web browser version is limited, and that the proper IDE, or downloadable platform, requires a paid subscription. On the other hand, Python is free and more widely used. It has the same modelling capacities as MATLAB, although perhaps less intuitive in that regard (since it is more general purpose). All things considered, it made more sense to develop the Dynamic Brownian Simulation in Python

than to reimplement the FEM solver in MATLAB.

To do that, it was desirable to find someone who was familiar with Python and who was interested in comparing the results of this Brownian Simulation with the old MATLAB model and with the other Firedrake model. Finally, a fresh perspective on the code and mathematical models would hopefully lead to new insights.

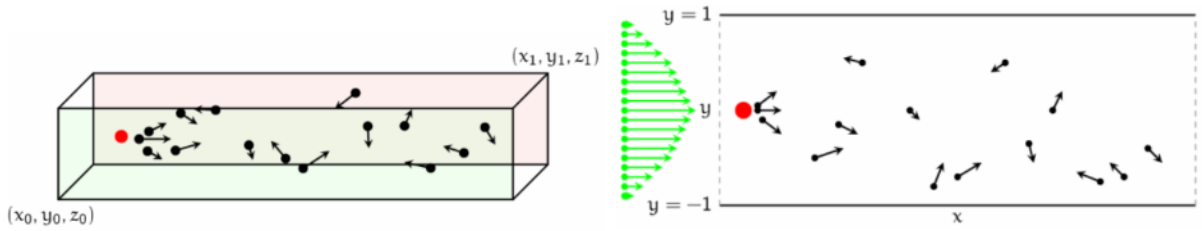
I had studied Scientific Computing (CS319) with Niall Madden this year and enjoyed the projects there. Additionally, I had heard of the Summer Bursary and was hoping to find a practical project with an element of coding that also built upon my Mathematical Science studies. After discussing Niall's available projects with him, we settled on this project. I would code the Dynamic Brownian Simulation in Python after taking the time to understand the model and also becoming familiar with the MATLAB version of it.

The interesting thing about this project is that no real world data is required to code it. Clearly, Niall Madden was able to suggest relevant values so I could test results of my Python version with the previous MATLAB version. However, downloading data files was not required.

Dynamic Brownian Motion computes a *numerical* solution to the equations describing the model. It's effectiveness in comparison to the finite element method (i.e., the Firedrake model for solving the advection-diffusion equation) can be tested using several typical values in the code for both models. However, this is a testament to the great practicality of this project. This same model, which I will unpack later, can be used for a variety of situations, from tracking pollutant particles in a channel of pure water to following the progress of drug injected into the bloodstream of a person.

2 Visualising Brownian Dynamic Motion in this project

To understand how a particular fluid A behaves when inserted in some channel containing another fluid B, it is best to recall that all fluids are made of many small particles. By initially treating fluid A as a specified number of particles, it is easier to predict the motion of fluid A while in fluid B. One such method of determining this motion is using stochastic differential equations, provided that the particles have Dynamic Brownian Motion. That is, the particles move in the direction of the fluid's current (downstream) while also oscillating randomly due to collisions with other particles (see Figure 1 where the current flows from left to right). Evidently, this applies to our model.



(a) 3-D model (b) 2-D model

Figure 1: 2-D channel is bird's eye view of 3-D channel with Brownian Dynamic Motion

Recalling that this model can be applied to industrial, environmental, or biological flow, for simplicity, I have chosen one particular version of the model to explain the mathematical problem

Table 1: Parameter symbols and typical values

Parameter	**Symbol**	**Range considered**
Péclet number	Pe	100
Schmidt number	Sc	10 – 1000
Lower-wall absorption strength	Γ_0	0.3
Upper-wall absorption strength	Γ_1	0.7
Oscillation amplitude	ε	1.0
Oscillatory Reynolds number	α	1.0

at hand and the output of the associated code. That is, pollutant particles (fluid A) entering a channel of pure water (fluid B). Although this is a 3-D situation, it turns out that the z-axis and the y-axis can be used interchangeably. Hence, it can be simplified to a 2-D problem with only an x-axis (horizontal or length of channel) and a y-axis (vertical or width of channel). Subsequent diagrams, i.e. Figure 2 onward, are generated by the Python code that I made during this project. These 2-D diagrams are looking down at the channel of pure water, like a bird's eye view. The upper boundary is actually the left bank and the lower boundary represents the right bank, if one was looking downstream. The fluid in the channel moves from left to right. Pollutant particles are entered at a fixed point, usually (0,0), and move left and right (represented as up and down in the diagrams).

To be specific, the scenario depicted in Figure 1 is of a fluid (water, for example), flowing through a channel. In addition, particles (represented by dots) are being carried by that flow. The goal of the model is to predict the concentration of the particles at any point in time and space.

The velocity of the fluid is constant in the x-direction, but varies both y and time, and is given as

$$u(y,t) = u_0(y) + u_1(y,t)$$

where

$$u_0(y) = \frac{1}{2}(1-y^2), \qquad \text{and} \qquad u_1(y,t) = -\Re\left[\frac{i\epsilon}{\alpha}\left(1 - \frac{\cosh\sqrt{i\alpha}y}{\cosh\sqrt{i\alpha}}\right)e^{i\alpha\,\mathrm{Sc}\,t}\right].$$

Note that $u(y,t)$ is the overall velocity, $u_0(y,t)$ is the velocity independent of time t and $u_1(y,t)$ is the velocity dependent on t. In addition $\Re(\cdot)$ is the real part of the complex-valued quantity $u_1(y,t)$. The other terms used in the model are summarised in Table 1, along with typical values.

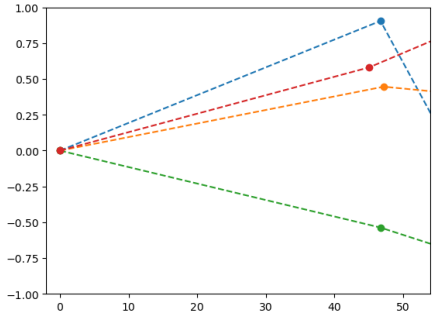
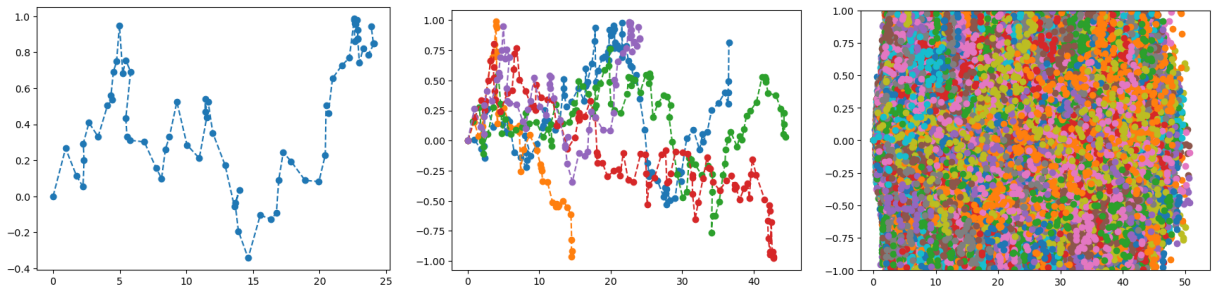


Figure 2: Trajectories of four sample pollutant particles shortly after they enter the channel

3 From abstraction to reality

Below is demonstrated sample particle movement over time for a single particle, 5 particles, and 2^7 particles.



(a) 1 particle (b) 5 particles (c) 2^7 particles

Figure 3: Bird's eye view of the channel as with different number of particles

Clearly, continuing in this manner will result in chaotic and time-consuming simulations. Therefore, although it is interesting to know how the pollutant particles move individually, ultimately it is more useful to know the concentration of these particles in the channel of fluid at a given time. By calculating the centre of mass of these particles, the associated behaviour and motion of the centre of mass can be determined. This will be further discussed in Section 5.

4 Summary of tasks completed during this internship

Initially, I took the time to become familiar with some key tools and online platforms including Git, a version control system, and Overleaf, a web-based online LaTeXplatform. I also became accustomed to MATLAB, particularly by analysing the MATLAB code of Niall Madden and Nanda Poddar for the Brownian Dynamic Solver.

Next I revised some key modules and functions in Python like `plt.plot()` for plotting graphs and `np.nan` for setting values to invalid.

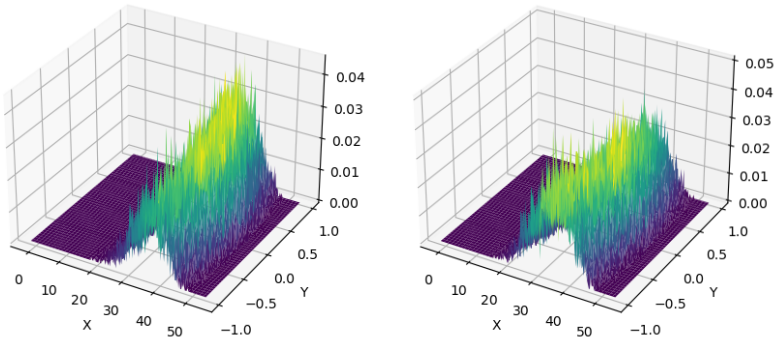
After that, I began coding in Python a basic version of the MATLAB code which had very few functions and which only had absorption of the particles at the boundaries, not reflection. Furthermore, I was able to use the same values for `Np` (number of particles) in the MATLAB code to ensure that the Python output was correct.

With the basic version of the code completed, I then analysed the motion of the particles. For example, by making different plots of the locations of the particles over different times, recording how many particles were still alive at the end of the simulation, introducing reflection at the boundaries and noting at which boundary the particles were absorbed or reflected.

Finally, I sought to make the code more efficient by introducing functions, later combining some of them, reducing the number of *if* statements and implementing masking. Timing the code before and after these modifications determined which parts of the code needed to be simplified.

5 Results and Analysis

By adjusting the probabilities of a particle being absorbed by the left (Γ_1) or right (Γ_0) boundaries, the location of the centre of mass of the pollutant particles (represented as yellow, i.e. the largest concentration of particles) changes:



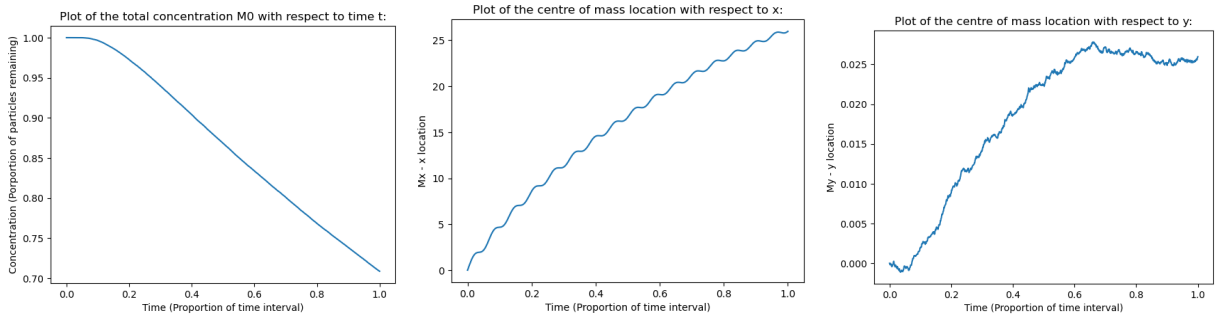
(a) $\Gamma_0 = 0.9$ and $\Gamma_1 = 0.1$ (b) $\Gamma_0 = 0.05$ and $\Gamma_1 = 0.95$

Figure 4: Showing the affect of Γ_0 and Γ_1 on boundary absorption

There are three key ways of confirming whether my Python code realistically portrays the movement of the pollutant particles. Firstly, it was expected that the total concentration of pollutant particles would consistently decrease over time, as a fixed number of pollutant particles was used. This is clearly demonstrated by Figure 5a.

Secondly, the x-location of the centre of mass should increase steadily with small consistent oscillations due to the fluid oscillating. This is evident in Figure 5b.

Finally, the y-location of the centre of mass should reflect the (Γ_0) and (Γ_1) chosen. In Figure 5c, Γ_0 is quite large and, hence, there is a high probability of particles being absorbed upon collision with the right boundary. Additionally, Γ_1 is small, and more particles are reflected off the left boundary upon collision. This results in a higher concentration of pollutant particles near the left boundary.



(a) Plot of M_0 (b) Plot of M_x (c) Plot of M_y

Figure 5: Looking at the centre of mass of the pollutant particles

6 Analysis on efficiency of the Python code

Through research and follow up conversations with Niall Madden, I discovered many new ways to code more effectively and efficiently. Here are a few interesting findings:

`np.random.default_rng(1)` is better than `np.random.random()` for generating random numbers because the former uses PCG64 which is higher quality and faster. However, the latter uses Mersenne Twister which is slower and flawed.

rng.standard_normal() is faster than *rng.normal*() as the customizable nature of the latter makes it slower if using the default values for the normal distribution. It performs additional steps in its algorithm to allow for the values to be customized. However, I was using the default values for the normal distribution and hence this was unnecessary.

7 Possible improvements to, and uses of, this project

All research and code can be improved and deepened. This project is no exception. Firstly, further analysis can be done with the code developed in this project and the FEM-solver.

Additionally, it would be nice to develop the code as a live simulation rather than a specified static frames.

Some possible uses of the code developed in this project include:

- Analysis of the best values of Pe, Γ_0 and Γ_1
- Discover the most effective place to put absorbing materials along the boundaries given Γ_0 and Γ_1
- Adjust Γ_0 and Γ_1 so that k % of the particles are always after m % of the fixed time interval

8 Conclusion

I thoroughly enjoyed this opportunity to combine various aspects of the mathematical sciences in a practical way. It was interesting to contribute to mathematical research with such broad real-world applications. Not only has this experience built my confidence to present mathematical findings in a professional setting, but I have also learned to use a variety of tools such as LaTeX.

I am very grateful to Niall Madden for supporting me in this project. I would also like to thank Neill O'Leary and all who organise this program for making this collaboration possible. I look forward to continuing to apply these skills to other projects in my coming final year.

9 List of resources I used

Shan Huang, et al. Transient dispersion of reactive solute transport in electrokinetic microchannel flow. *Physics of Fluids* 1 May 2024; 36 (5): 052011. https://doi.org/10.1063/5.0206129

Ferziger, J. H., Perić, M., & Street, R. L. (2020). *Computational Methods for Fluid Dynamics* (4th ed.). Springer.

Allen, L. J. S. (2011). *An Introduction to Stochastic Processes with Applications to Biology* (2nd ed.). CRC Press.

IBM Technology. (2022, August 11). *What is Monte Carlo Simulation ?* [Video]. YouTube. https://youtu.be/7TqhmX92P6U

Decision Lab. (2025, June 5). *Monte Carlo Simulation Explained in 5 min* [Video]. YouTube. https://youtu.be/psOYFdx838E

Uplatz. (2026, January 4). *The Monte Carlo Method: Origins, Theory, and Practical Applications* [Video]. YouTube. https://youtu.be/p9Z7xppLRSw

FuseSchool - Global Education. (2013, May 30). *What Is Brownian Motion?* [Video]. YouTube. https://youtu.be/4m5JnJBq2AU

SciToons. (2023, September 18). *Brownian Motion: Explaining Life's Randomness* [Video]. YouTube. https://youtu.be/z6vwS0fue6E

My Book of Science. (2021, March 16). *Brownian Motion and Einstein? What is Brownian Motion? (feat. Yoda)* [Video]. YouTube. https://youtu.be/i7tQLjGZR0A

hUndefined. (2025, April 22). *How to Install Git on Windows — Full Guide* [Video]. YouTube. https://youtu.be/VWbX2B3Q-1g

Alex The Analyst. (2025, September 2). *Installing and Setting up Git and GitHub* [Video]. YouTube. https://youtu.be/wDRoduig_98

Overleaf. (n.d.). *Paragraphs and new lines.* https://www.overleaf.com/learn/latex/Paragraphs_and_new_lines

Overleaf. (n.d.). *Inserting Images.* https://www.overleaf.com/learn/latex/Inserting_Images%23Positioning

Overleaf. (n.d.). *Hyperlinks.* https://www.overleaf.com/learn/latex/Hyperlinks

LaTeX-Tutorial.com. (n.d.). *Insert an image in LaTeX - Adding a figure or picture.* https://latex-tutorial.com/tutorials/figures/

The MathWorks, Inc. (2023, July 31) *What are anonymous functions in MATLAB, and how can they be used to simplify code? Provide an example where an anonymous function is used effectively.* https://www.mathworks.com/matlabcentral/answers/2002742-what-are-anonymous-functions-in-matlab-and-how-can-they-be-used-to-simplify-code-provide-an-exampl

SCALER. (2022, May 25). *Length of Array in Python.* https://www.scaler.com/topics/length-of-array-in-python/

W^3 schools. (n.d.). *Python Lambda.* https://www.w3schools.com/python/python_lambda.asp

W^3 schools. (n.d.). *Python cmath Module.* https://www.w3schools.com/python/module_cmath.asp

W^3 schools. (n.d.). *Python time Module.* https://www.w3schools.com/python/ref_module_time.asp

W^3 schools. (n.d.). *Python Add Array Item.* https://www.w3schools.com/python/gloss_python_array_add.asp

W^3 schools. (n.d.). *NumPy Creating Arrays.* https://www.w3schools.com/python/numpy/numpy_creating_arrays.asp

W^3 schools. (n.d.). *Python While Loops.* https://www.w3schools.com/python/python_while_loops.asp

GeeksforGeeks (2025, July 23). *Python program to Return the real part of the complex argument.* https://www.geeksforgeeks.org/python/python-program-to-return-the-real-part-of-the-complex-argument/

GeeksforGeeks (2025, July 23). *How to Do a Scatter Plot with Empty Circles in Python.* https://www.geeksforgeeks.org/python/how-to-do-a-scatter-plot-with-empty-circles-in-python/

GeeksforGeeks (2026, February 15). *NumPy - linspace() Function.* https://www.geeksforgeeks.org/python/numpy-linspace/

GeeksforGeeks (2024, July 26). *Random Numbers in Python* https://www.geeksforgeeks.org/python/random-numbers-in-python/

GeeksforGeeks (2018, February 9). *Graph Plotting in Python — Set 3* https://www.geeksforgeeks.org/python/graph-plotting-python-set-3/

References

[1] David A. Ham, Paul H. J. Kelly, Lawrence Mitchell, Colin J. Cotter, Robert C. Kirby, Koki Sagiyama, Nacime Bouziani, Sophia Vorderwuelbecke, Thomas J. Gregory, Jack Betteridge, Daniel R. Shapero, Reuben W. Nixon-Hill, Connor J. Ward, Patrick E. Farrell, Pablo D. Brubeck, India Marsden, Thomas H. Gibson, Miklós Homolya, Tianjiao Sun, Andrew T. T. McRae, Fabio Luporini, Alastair Gregory, Michael Lange, Simon W. Funke, Florian Rathgeber, Gheorghe-Teodor Bercea, and Graham R. Markall. *Firedrake User Manual.* Imperial College London and University of Oxford and Baylor University and University of Washington, first edition edition, 5 2023.

[2] The MathWorks Inc. Matlab version: 26.1.0.3203278 (r2026a), 2026.

[3] Gourab Saha, Nanda Poddar, Kajal Kumar Mondal, and Niall Madden. Hydrodynamic dispersion of volatile contaminant in an open channel flow using a fitted operator approach. In *Proceedings of the 2nd International Conference on Nonlinear Dynamics and Applications (ICNDA 2024)*, volume 315 of *Springer Proceedings in Physics*. Springer, 2024. https://doi.org/10.1007/978-3-031-69134-8_47.