

UNIVERSITY OF SOUTHAMPTON

FEEG3003 INDIVIDUAL PROJECT

Analysis of Droplet Evaporation for Large Particle Populations Using GPUs and the OpenCL Language

by:
Andrew Kernan

Supervisor:
Prof. John SHRIMPTON

Word count: 7582

This report is submitted in partial fulfillment of the requirements for the MEng Aeronautics and Astronautics, Faculty of Engineering and Physical Sciences, University of Southampton

April 10, 2020

Declaration

I, Andrew Kernan declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;
2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. None of this work has been published before submission.

Acknowledgements

I would like to thank my supervisor Professor John Shrimpton for his guidance and help during the project.

I would also like to thank Elijah Andrews for taking the time to explain and fix bugs in his code.

Contents

Abstract	7
Nomenclature	8
1 Literature Review	11
2 Background Material	13
2.1 GPU Computing	13
2.2 Physics of Droplet Evaporation	14
2.2.1 Heat Transfer Processes	14
2.2.2 Vapour Pressure	14
2.3 Mass Transfer	15
2.3.1 Droplet Physical Model	15
3 Droplet Evaporation Model	17
3.1 Definition of Models	17
3.2 Assumptions	17
3.3 Governing Equations	18
3.3.1 Particle Timescales	18
3.3.2 Position	18
3.3.3 Velocity	18
3.3.4 Temperature	20
3.3.5 Mass	20
4 Fluid Model	21
4.1 Taylor-Green Vortex	21
5 Numerical Methods	23
5.1 Timestepping Procedure	23
5.2 Euler Method	24
5.3 Modified Euler Method	25
5.4 Runge-Kutta Method	25
5.5 Backward Euler Method	26

6	Single Particle Solution Method	28
6.1	Simulation Settings	28
6.2	Uncoupled Heat Transfer	28
6.3	Uncoupled Mass Transfer	31
6.4	Coupled Heat and Mass Transfer	40
7	Verification and Validation of the Existing Code	45
7.1	Analytic Test Case	45
7.2	Statistical Test Case	47
7.2.1	Run Time Test	50
7.3	Problems Encountered with the Existing Code	51
7.3.1	Periodic Boundary Condition	51
7.3.2	Memory Alignment of Structures	52
7.3.3	File Directory Checking	52
7.3.4	Initialisation of Particle Speeds	53
7.3.5	Logsteps Being Skipped	54
8	OpenCL Implementation	55
8.1	Additions made to the Code	55
8.1.1	Source Code Structure	55
8.1.2	Variables	57
8.1.3	Heat and Mass Transfer Model	57
8.2	Verification	57
8.2.1	Uncoupled Heat Transfer	57
8.2.2	Uncoupled Mass Transfer	57
8.2.3	Coupled Heat and Mass Transfer	57
9	Results	58
10	Conclusion	59
10.1	Summary of Results	59
10.2	Future Work	59
10.2.1	Additional Models and Droplet Species	59
10.2.2	Re-ordering of Code to Simplify Running Simulations	59

10.2.3 Quantitative Analysis and Optimisation of the Code	59
A Method Statement	60
B Derivations	60
B.1 Velocity of a Particle under Weight and Stokes Drag Forces	60
B.2 Backward Euler Method for Temperature ODE	61
B.3 Backward Euler Method for Mass ODE	61
B.4 Uncoupled Heat Transfer	61
B.5 Uncoupled Mass Transfer	62
B.6 Terminal Velocity	63
C Physical Data	63
C.1 Air	64
C.2 Water	64
D Existing Codebase Structure	65
References	68

Abstract

The simulation of large populations of particles is computationally an expensive process. Simply running the code on CPUs does not make sense as the simulation times do not scale with the number of particles. GPUs, however, have an architecture that makes them a superior compute device compared to CPUs for highly parallel operations.

This project will implement a commonly used model for heat and mass transfer, first in Python to develop and evaluate solving methods. With the final simulation code being written in C and OpenCL. A number of simulations with large populations of particles will be run and statistics on the particles calculated.

Findings

Summary

Nomenclature

Arabic Symbols

\bar{R}	Universal gas constant	$K \text{ kg mol}$
$B_{m,eq}$	Spalding transfer number for mass	—
C	Molar concentration	mol/m^3
D	Diameter	mm
d	Diffusion coefficient	cm^2/s
f_1	Correction factor for Stokes drag	—
f_2	Correction factor for heat transfer due to evaporation	—
g_i	Acceleration due to gravity	$9.81 \text{ m}/\text{s}^2$
H_M	Specific driving potential for mass transfer	—
$H_{\Delta T}$	Collection of terms contributing to non-uniform internal temperature effects	—
J	Diffusive flux density	$\text{mol}/\text{m}^2/\text{s}$
L_V	Latent heat of evaporation	J/kg
m	Mass	kg
P	Pressure	Pa
T	Temperature	K
u_i	Carrier gas velocity	m/s
v_i	Droplet velocity	m/s
W	Molecular weight	$\text{kg}/\text{kg mole}$
X_i	Transient position	m
Y_G	Free stream vapour mass fraction	—
$Y_{s,eq}$	Vapour mass fraction at the droplet's surface	—

Greek Symbols

$\chi_{s,eq}$	Surface equilibrium mole fraction	—
Γ	Binary diffusion coefficient	m^2/s
λ	Thermal conductivity	$\text{J m}^{-1} \text{s}^{-1} \text{K}^{-1}$
μ	Viscosity	$\text{kg}/\text{m}/\text{s}$
ρ	Density	kg/m^3
τ_d	Particle time constant for Stokes flow	s
θ_1	Ratio of heat capacity for constant pressure between the gas and liquid phase	—

θ_2 Ratio of molecular weights

—

Subscripts

0 Value of property at start of simulation

B Property boiling

C Carrier gas property

d Droplet property

eq Property in equilibrium

G Gas phase property

i Vector component

n Value of property at current time level

$n + 1$ Value of property at next discrete time level

s Property at droplet surface

sat Property at saturation

Superscripts

P Placeholder

Non-dimensional Numbers

Le Lewis Number

Nu Nusselt Number

Pr Prandtl Number

Re Reynolds Number

Sc Schmidt Number

Sh Sherwood Number

Acronyms

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

GPU Graphics Processing Unit

IP Individual Project

MPI Message Passing Interface

ODE Ordinary Differential Equation

OpenCL Open Computing Language

Introduction

This individual project builds off an individual project from two years ago titled “GPU Enabled Analysis of Agglomeration in Large Particle Populations”. The goal of that project was to simulate up to 10^7 particles in a fluid including particle collisions, to observe how agglomerates form. As well as to vary the simulation properties and to do statistical analysis of the resulting agglomerate properties [1].

This individual project is titled “Analysis of Kerosene Particle Evaporation for Large Particle Populations Using GPUs and the OpenCL language”. With the aim to simulate the evolution of heat and mass transfer for 10^6 particles using GPUs. The objectives required for this are:

1. Run the previous code and test it.
2. Compute coupled heat and mass transfer of a single Kerosene particle.
3. Run large scale simulations (1 million particles). This could lead into post-processing of the results. For example, generating probability density functions.

Objective 1 is imperative as the code produced from objective 2 will be added into the main time loop in the previous code. Objective 1 and 2 can run in parallel though; the aim is to first develop a suitable numerical model for simulating the heat and mass transfer of the particle in Python. This does not require the previous project’s code, although the final code required for the large scale simulations will be required to work with the existing code. This approach reduces the risk of the project timelines slipping. Because if progress is slower than expected on one objective, more focus can be given to the other objective, allowing the project to continue to advance. Objective 3 requires both objective 1 and 2 to be completed, the post-processing of results etc. . . . is a stretch goal for the project if time permits.

This report first introduces literature fundamental to the project and reviews the content of the literature. As well as explain the fundamental maths and physics that governs the simulation of the particles. The next section evaluates methods for implementing the models.

1 Literature Review

As previously mentioned this IP builds off a previous IP titled “GPU Enabled Analysis of Agglomeration in Large Particle Populations”. The project developed simulation code in Python and OpenCL for simulating the collision and agglomeration of particles. This included a solution procedure for iterating the position and velocity of the particles [1]. This project uses the code produced by this previous project but removes the Discrete Element Method (DEM) model for the collisions as this is computationally expensive.

The key paper for this project with regards to simulating the heat and mass transfer processes is titled “Evaluation of equilibrium and non-equilibrium evaporation models for many-droplet gas-liquid flow simulations” by Miller, Harstad and Bellan. This paper is significant in that it details and compares eight different droplet evaporation models based on the same fundamental Lagrangian equations for droplet position, velocity, temperature and mass [2]. The eight models in the paper are denoted by Mx where x is the model number. As will be justified in Section 3 the model of choice for this project is M1 which is a form of the classical evaporation the earliest model known as the infinite conductivity model.

[2] is an unusually long paper although it presents some results that will be useful in validating the simulation code. Of note is Figure 2 with $Re = 0$ which means terms involving convection (particularly in the Nusselt and Sherwood numbers) can be neglected. Which provides a simpler test case. Moderate gas and droplet temperatures were also used again making the results simpler. There does appear to be a typo however in the caption, the starting droplet diameter D_0 was reported to be 1.1 mm contradicting the data in the plot with D^2 at $t = 0$ being 1.1 mm^2 . For using this test case later in the report it is assumed $D^2 = 1.1\text{ mm}^2$ with D_0 being 1.05 mm . This typo does not appear elsewhere in the paper, however. Although a key frustration in a paper that presents so many models is it can be hard to decipher how certain properties are evaluated. This issue is highlighted in a paper from the Center for Turbulence Research that evaluated how the definition of such properties affects the simulation sensitivity [3].

One of the most recent citations uses models M1 and M2 to simulate the evaporation of fuel particles in flame combustion. However, it used $\frac{\beta}{e^{\beta}-1}$ instead of 1 for one of the parameters f_2 [4]. Which as noted in [2] was not considered commonly used at the time that paper was published. The paper also investigates the effect of setting the Lewis number, Le to 1. It found that overall this assumption only increased the evaporation rate on the M2 model slightly. Although the effect was found to be more prominent for forced convection cases. In terms of the performance of the models the M1 model was found to more closely match experimental data for the forced convection of hexane. With $d_{p,0} = 1.76\text{ mm}$, $T = 437\text{ K}$ and $Re_0 = 110$. The same test for decane found that both models deviated from the experimental results but the M2 model was more accurate. For the decane case the conditions where $d_{p,0} = 2.0\text{ mm}$, $T = 1000\text{ K}$ and $Re_0 = 17$. The evaporation was for droplets in air. The experimental results are from [5], the same as those used in [2]. This is a good indication the experimental data used in [2] is still relevant.

The models only form half of the solution with regards to simulating populations of particles. Equally of importance is the method by which the equations are to be solved. It has already been determined from [1] the position and velocity equations are best solved numerically with an implicit method that gives higher than first order accuracy. A number of papers in the literature have used fourth order Runge-Kutta methods for solving the heat and mass equations [2] [6] [7]. Older papers such as “A Comparison of Vaporization Models in Spray Calculation” by Aggarwal et al. use a second order Runge-Kutta method

[8].

Examples of GPU simulation code in the literature seemed to be less common.

Aforementioned papers have used CPU codes as the Lagrangian simulation of the droplet is normally coupled with a CFD code for the fluid. GPUs have been used to accelerate the particle Lagrangian equations, [9] provides an in-depth analysis of this. The paper highlights the Lagrangian code is accelerated by up to 14 times compared with the original CPU code. Where GPUs have been used CUDA seemed to be the language of choice [9] [10]. There have been studies using the Boltzmann Lattice Method that have used OpenCL [11] [12]. In terms of an implementation of the droplet evaporation model of the type presented here no implementations using OpenCL were found in the literature.

2 Background Material

2.1 GPU Computing

The question, why use GPUs? Naturally arises from the title of this project. CPUs in large computing clusters like the University of Southampton's Iridis Compute Cluster could be used. This would simplify the project as extra code called a kernel has to be written for the simulation to run on a GPU. The problem arises in the way CPUs share data in computing clusters. To write code for CPU parallel computing MPI has to be used. A block diagram of this is shown in Figure 2.1.

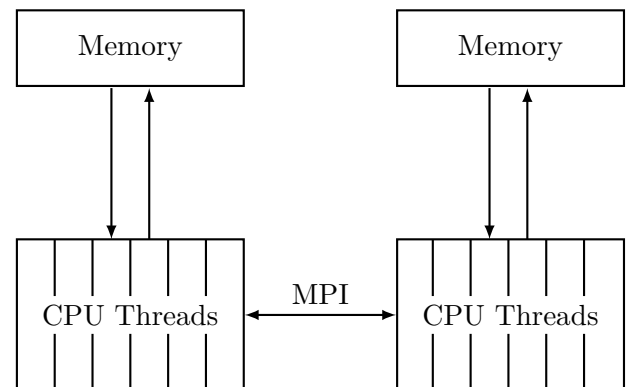


Figure 2.1: MPI block diagram.

This process of sharing data between compute nodes is what really limits CPU parallel computing. The overhead this adds would cause simulations not to scale linearly compared with using a GPU for a highly parallel task. As by comparison the GPU threads are able to access the same shared memory as shown in Figure 2.2. With regards to writing GPU kernels there are two languages to choose from. The Open Computing Language, OpenCL and the Compute Unified Device Architecture, CUDA. CUDA is a language developed by Nvidia that only runs on Nvidia GPUs, whereas OpenCL is open source and can run any GPU and indeed a wider range of compute devices. The existing code uses OpenCL kernels, to enable greater hardware compatibility, so OpenCL is the language that will be used for this project.

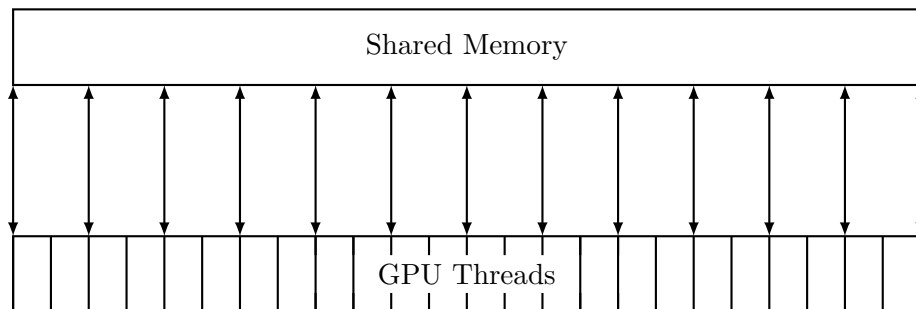


Figure 2.2: GPU memory block diagram.

2.2 Physics of Droplet Evaporation

2.2.1 Heat Transfer Processes

Fundamental to modelling droplets is an understanding of multiphase fluids. Droplet laden flows, (the subject of this report) are an example of two phase flows.

With regards to the physics of droplet evaporation there are two fundamental driving processes. These are diffusion and convection.

Diffusion is a process where there is a net movement of particles from a region of high concentration to a region of low concentration. This is driven by the concentration gradient so is affected by properties of the particles and the distance over which diffusion occurs. The carrier species of the particles will also affect the flux. Fick's Law quantifies this as:

$$J(x, y, z) = -d \left(\frac{\partial C}{\partial x}, \frac{\partial C}{\partial y}, \frac{\partial C}{\partial z} \right) \quad (2.1)$$

[13].

More useful is thermal diffusion, with the rate defined by Fourier's Law. For example, in one dimension as:

$$\dot{q}_x = -k \frac{dT}{dx} \quad (2.2)$$

With x in the direction of heat transfer, \dot{q}_x the heat transfer rate per unit area, k the thermal conductivity and dT/dx the temperature gradient in the x -direction. [14]

Convection transfers heat energy in the direction of motion of the fluid. Convection can be forced (some fluid flow forcing convective transfer), or natural (the flow is created by the forces resulting from the heat transfer). Convective heat transfer for a fluid is given by:

$$\dot{q} = h\Delta T \quad (2.3)$$

where ΔT is the difference between the temperature at the surface of the fluid and the temperature at infinity: $T_s - T_\infty$. The convective heat transfer coefficient h contains variables that allow for a simple form for the convective heat transfer equation.[15]

In general an exact solution for h is only available for specific cases such as laminar flow. More common is to form an empirical relation using non-dimensional numbers that capture the physics of the problem. [16]

2.2.2 Vapour Pressure

The driving force for changes to the properties in the vapour phase is the vapour pressure. Assuming the droplet is formed of a pure liquid species the vapour pressure is only a function of temperature. For evaporation to occur the partial pressure of the liquid must be less than the vapour pressure [13].

The vapour pressure can be found by considering that for two phases to be in equilibrium their respective chemical potentials, μ_{pot} must be equal. This implies here that $\mu_{liquid} = \mu_{gas}$ and any changes must be equal $\rightarrow d\mu_{liquid} = d\mu_{gas}$. If there is a change of pressure acting on the liquid phase, i.e. $dV_{m,liquid}dP = d\mu_{liquid}$. (With $dV_{m,liquid}$ the molar volume of the liquid phase). Therefore, $dV_{m,gas}dP_{vapour} = d\mu_{gas}$, with dP_{vapour} the change in vapour pressure. Equating the changes in the liquid and gas phase, it is possible using

integration and assuming the gas is perfect and the molar volume of gas is much greater than the molar volume of the liquid to derive the Clausius-Clapeyron equation. [17]

$$\chi_{s,eq} = \frac{P_{atm}}{P_G} \exp \left[\frac{L_V}{\bar{R}/W_V} \left(\frac{1}{T_B} - \frac{1}{T_d} \right) \right] \quad (2.4)$$

So the equilibrium mole fraction of the vapour can be found.

2.2.3 Mass Transfer

2.2.4 Droplet Physical Model

The droplet consists of a liquid phase surrounded by a vapour phase, with the droplet surrounded by a gas phase which is the carrier gas species. A diagram of the physical model of the droplet is shown in Figure x.

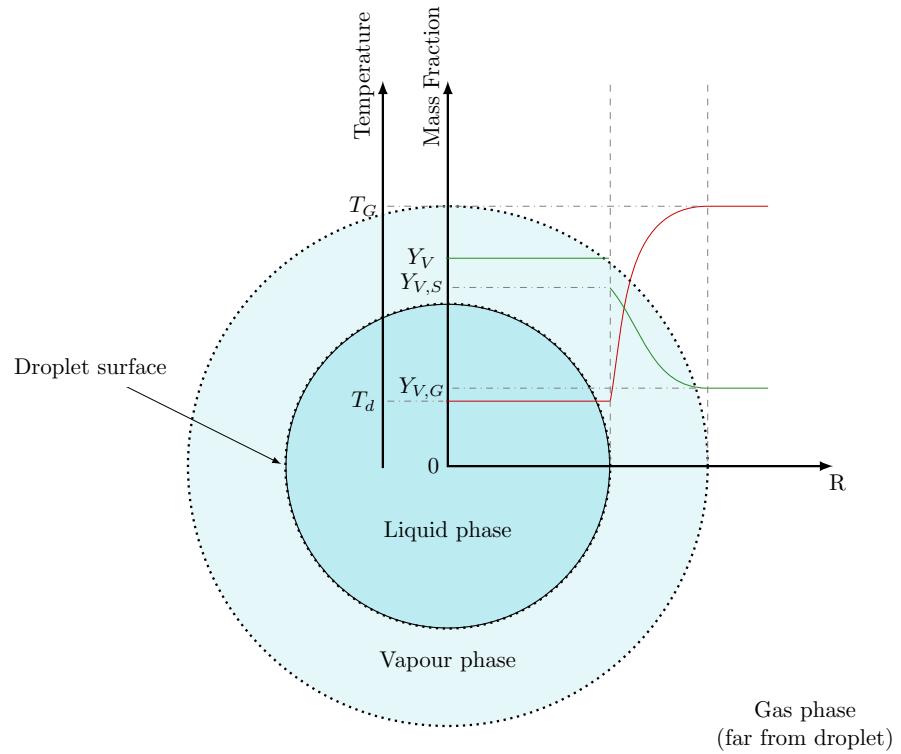


Figure 2.3: Diagram of how the droplet's and carrier gas phases with their associated properties vary with droplet radius.

The droplet temperature is assumed to be uniform within the droplet, although various with time. This. In reality there is a thermal gradient across the droplet. The internal temperature profile can be modelled as part of M8 in [2]. The temperature rises across the vapour phase to the gas temperature.

Similarly within the droplet the mass fraction of droplet vapour within the droplet is uniform and is equal to 1. The mass fraction drops instantaneously at the surface, as the surface of the droplet is a mixture of droplet vapour and gas phase. The mass fraction of vapour decreases across the vapour phase until an equilibrium mixture with the gas phase is reached.

[18].

3 Droplet Evaporation Model

3.1 Definition of Models

The 8 models in [2] are denoted by Mx where x is the model number.

What defines the models are the variables f_1 , f_2 , Nu , Sh , $H_{\Delta T}$ and H_M . Although the main variation in the models stems from f_2 , $H_{\Delta T}$ and H_M . M1 is known as the classical rapid mixing model. M2 will be ruled out for the purposes of this project as the B'_T term in the evaporation correction f_2 must be solved iteratively at each timestep. This increases the computation overhead of the simulation which may be acceptable for a single droplet but will be problematic for large population of droplets.

Model	Name	f_2	$H_{\Delta T}$	H_M
M1	Classical rapid mixing	1	0	$\ln [1 + B_{M,eq}]$
M2	Abrmazon-Sirignano	$\frac{-\dot{m}_d}{m_d B'_T} \left[\frac{3Pr_G \tau_d}{Nu} \right]$	0	$\ln [1 + B_{M,eq}]$
M3	Mass analogy Ia	1	0	$B_{m,eq}$
M7	Langmuir-Knudsen I	$\frac{\beta}{e^{\beta}-1}$	$\frac{2\beta}{3Pr_G} \left(\frac{\theta_1}{\tau_d} \right) \Delta_s$	$\ln [1 + B_{M,neq}]$

Table 3.1: Detail of variables used in candidate models.

A more suitable model to provide a point of comparison in this case would be one of the mass analogy models, M3-M6. Which are derived from vapour mass fraction boundary condition at the droplet's surface. This assumes the droplet does not dissolve in the gas phase. These models use the same formulations for the surface vapour mass fraction (Y_s), Spalding transfer number for mass (B_m) and mass transfer potential ($H_{\Delta T}$). This would simplify producing code for an extra model.

Models M7 and M8, are to be ruled out as they use non-equilibrium formulations of the coefficients which would further complicate the code. The comparison between models is left as future work. With the choice of model for this project will be M1 due to its wide usage and ease of implementation

3.2 Assumptions

The model presented here is subject to the following assumptions:

1. The droplets are formed of a single species
2. The droplet is spherical and remains spherical for the entire evaporation process
3. There is no interaction between droplets
4. Within the droplet there is
 - (a) Zero temperature gradient
 - (b) Zero concentration gradient
5. The droplet temperature is homogeneous

6. The properties of the carrier gas are constant and are unaffected by the process of droplets evaporating

[2][19].

3.3 Governing Equations

Each of the models are defined by the same four Lagrangian equations for the droplets:

$$\frac{dX_i}{dt} = v_i \quad (3.1)$$

$$\frac{dv_i}{dt} = \left(\frac{f_1}{\tau_d}\right) (u_i - v_i) + g_i \quad (3.2)$$

$$\frac{dT_d}{dt} = \frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d}\right) (T_G - T_d) + \left(\frac{L_V}{C_L}\right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \quad (3.3)$$

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} \left(\frac{m_d}{\tau_d}\right) H_M \quad (3.4)$$

3.3.1 Particle Timescales

τ_d is the momentum relaxation timescale for the particle. This provides a timescale for each particle. It is determined from:

$$\tau_d = \frac{\rho_d D^2}{18\mu_G} \quad (3.5)$$

There is an equivalent timescale for the heat transfer process. This can be derived from equation 6.2 as:

$$\tau_T = \tau_d \left(\frac{3Pr_G}{Nuf_2\theta_1} \right) \quad (3.6)$$

These two timescales are exceptionally useful for the non-dimensionalisation of results. This provides a way of evaluating how quickly processes occur in the timescale of the particle. Which is a far more useful insight into the physics taking place than particle x takes y seconds to evaporate.

3.3.2 Position

Equation 3.1 and a simpler form of equation 3.2 were used in the IP this project is based off. The differential equation for position was solved using the trapezoidal rule, as this method has second order accuracy and the result is a linear first order equation:

$$X_{n+1} = X_n + \frac{v_n + v_{n+1}}{2} \Delta t \quad (3.7)$$

[1]

3.3.3 Velocity

For equation 3.2, the acceleration of the droplet is defined using Stokes' Law and can be derived by considering the forces acting on the particle:

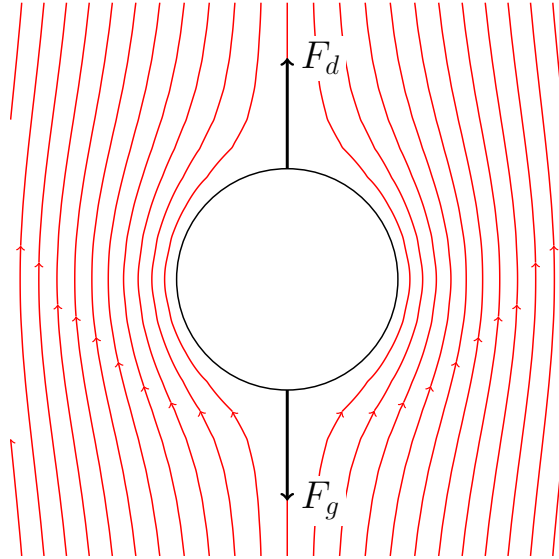


Figure 3.1: Forces acting on a particle under a Stokes flow regime.

The total force acting on the particle is:

$$F = F_g + F_d \quad (3.8)$$

This can be solved as a differential equation:

$$F = ma \quad (3.9a)$$

$$F = m \frac{du}{dt} \quad (3.9b)$$

$$\frac{du}{dt} = \frac{F(u)}{m} \quad (3.9c)$$

The force acting on the particle is the sum of the weight and drag forces. The weight force is given by:

$$F_g = mg \quad (3.10)$$

It is assumed the particle experiences a Stokes drag regime, therefore:

$$F_d = \frac{m}{\tau_d}(u_f - u) \quad (3.11)$$

So, the differential equation is:

$$\frac{du}{dt} = \frac{F(u)}{m} \quad (3.12a)$$

$$\frac{du}{dt} = \frac{F_g + F_d}{m} \quad (3.12b)$$

$$\frac{du}{dt} = \frac{mg + \frac{m}{\tau_d}(u - v)}{m} \quad (3.12c)$$

$$\frac{du}{dt} = \frac{1}{\tau_d}(u - v) + g \quad (3.12d)$$

This is for one vector component, hence the 3D case is:

$$\frac{dv_i}{dt} = \frac{1}{\tau_d}(u_i - v_i) + g_i \quad (3.13)$$

is dependent on three terms. Acceleration due to gravity g_i , and the difference between the carrier gas velocity and the velocity of the particle, $u_i - v_i$. The third term is a correction for Stokes drag that is dependent on the time constant for Stokes flow.

A formulation for f_1 is given in [2] as:

$$f_1 = \frac{1 + 0.0545Re_d + 0.1Re_d^{0.5}(1 - 0.03Re_d)}{1 + a|Re_b|^b} \quad (3.14)$$

With the Reynolds numbers defined as:

$$Re_d = \frac{\rho_G u_s D}{\mu_G} \quad (3.15a)$$

$$Re_b = \frac{\rho_G u_b D}{\mu_G} \quad (3.15b)$$

The velocities defined as:

$$u_s = |u_i - v_i| \quad (3.15c)$$

$$u_b = -\frac{\dot{m}}{\pi \rho_G D^2 u_b} \quad (3.15d)$$

And the constants a and b defined as:

$$a = 0.09 + 0.077 \exp(-0.4Re_d) \quad (3.15e)$$

$$b = 0.4 + 0.77 \exp(-0.04Re_d) \quad (3.15f)$$

Which is an empirical result produced from a data fit. In this format it is valid for $0 \leq Re_d \leq 100$ and $0 \leq Re_b \leq 10$. For the purpose of this research f_1 will be set to 1. This reduces the computation per timestep and still allows for comparisons to be made with data in the literature. As the models can be driven by iterating from an initial Reynolds number instead of using equation 3.2.

3.3.4 Temperature

The temperature ODE is a combination of a convective and diffusive processes. The convection is determined by:

$$\dot{Q} = \frac{Nu}{\tau_d}(T_G - T_d) \quad (3.16)$$

3.3.5 Mass

4 Fluid Model

As noted in the Literature Review the Lagrangian droplet model is normally coupled with a CFD code. Quite complex models could be used for this, for example, for investigations into the effect of turbulence. However, the focus here is getting a droplet evaporation model to be solved on a GPU and to investigate some statistics. Complex CFD codes could be added at a later date.

4.1 Taylor-Green Vortex

A set of analytic expressions for the fluid velocity will be used. The previous IP used a Taylor-Green vortex and that is what will also be used here. It is defined as:

$$U = A \cos\left(a\left(x + \frac{\pi}{2a}\right)\right) \sin\left(a\left(y + \frac{\pi}{2a}\right)\right) \cos\left(a\left(z + \frac{\pi}{2a}\right)\right) \quad (4.1a)$$

$$V = A \sin\left(a\left(x + \frac{\pi}{2a}\right)\right) \cos\left(a\left(y + \frac{\pi}{2a}\right)\right) \sin\left(a\left(z + \frac{\pi}{2a}\right)\right) \quad (4.1b)$$

$$W = -2A \sin\left(a\left(x + \frac{\pi}{2a}\right)\right) \sin\left(a\left(y + \frac{\pi}{2a}\right)\right) \cos\left(a\left(z + \frac{\pi}{2a}\right)\right) \quad (4.1c)$$

With A defined as the flow magnitude and a defined as the vortex frequency. This flow field is visualised in Figure 4.1.

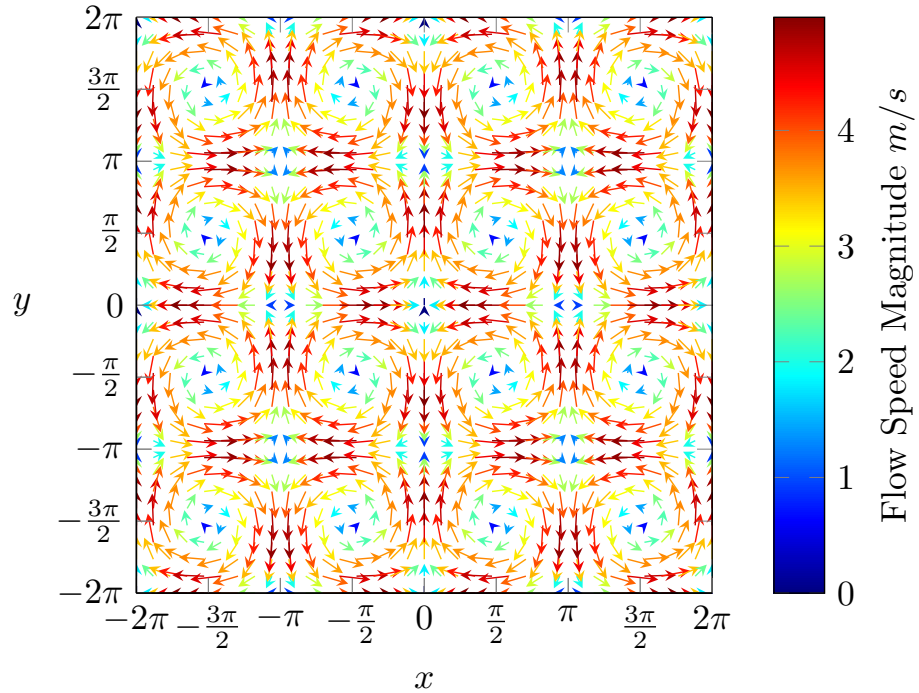


Figure 4.1: A 2D xy vector field of a Taylor-Green vortex. Using $A = 5$, $a = 1$ and a domain length of 4π .

The length of one of the individual vortices is the characteristic length l_0 used in defining the Stokes number.

$$Stk = \frac{\tau_{d0} u_0}{l_0} \quad (4.2)$$

Where:

$$\tau_{d0} = \frac{\rho_d D^2}{18\mu_G} \quad (4.3a)$$

$$u_0 = 0.7839A \quad (4.3b)$$

$$l_0 = \frac{\pi}{a} \quad (4.3c)$$

In this case $l_0 = \pi$ and $u_0 = 3.9195$ [1]. The usage of the Stokes number can be found in Section 7.2.

5 Numerical Methods

Given the governing equations are coupled and non-linear finding an analytical solution has not proved possible. It is therefore necessary to use a numerical scheme to solve the equations. The choice of scheme and its implementation are especially important when considering the solution must work for millions of droplets. The scheme should of course be accurate but its complexity is also a consideration. There is no point choosing a method that is very accurate but is slow because the because this will be compounded greatly across millions of particles. In addition to this the method must be stable and resistant to non-physical phenomena. (I.e. the droplet mass cannot go below zero).

In terms of implementation the order in which the equations are solved may prove to be important. For example at a given time level should the temperature of the droplet be found before the mass? And what kind of error checking within the time step should take place?

Analytical solutions to the equations decoupled are available and provide a point of comparison for some of the schemes.

5.1 Timestepping Procedure

Numerical methods work by taking discrete steps forward in time. The length of the simulation is determined by the size of the timestep and the maximum number of timesteps. The timestepping procedure is shown in Figure 5.1.

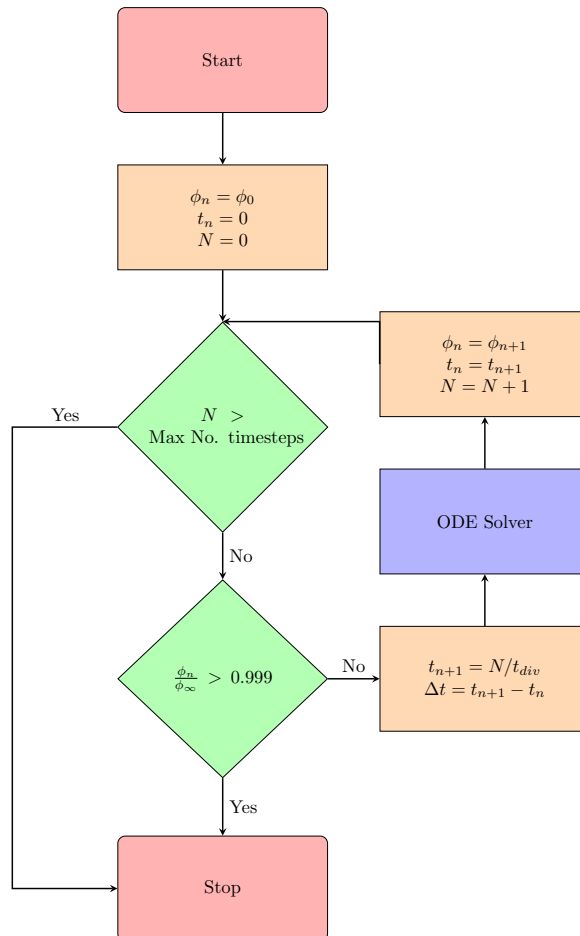


Figure 5.1: General timestepping procedure.

There are two conditions for the loop to run. The maximum number of time steps must not have been exceeded and the physical property ϕ cannot have converged to its value at infinity. This works for droplet temperature whilst for mass the condition would be a ratio of current to initial mass being greater than zero. Limiting the maximum number of time steps is necessary to prevent an infinite recursion in the case of the simulation being unable to converge. Also by specifying the value t_{div} in addition to N the size of time step and total time to be simulated is set.

The ODE solver is a black box that returns a numerical solution to the ODE at each discrete timestep. Some examples are given in the following subsections.

5.2 Euler Method

The simplest numerical method is to step forward in time and take the value at the next time level to be the current value plus a prediction on what the change between the values will be. This can be derived from the Taylor Series:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \dots + \frac{h^{n-1}}{(n-1)!}f^{(n-1)}(x) + \frac{h^n}{n!}f^n(c) \quad (5.1)$$

Using Δt as h and a as the value at the current time level n , from the first two terms in the series:

$$f(n+1) = f(n) + \Delta t f'(n) \quad (5.2)$$

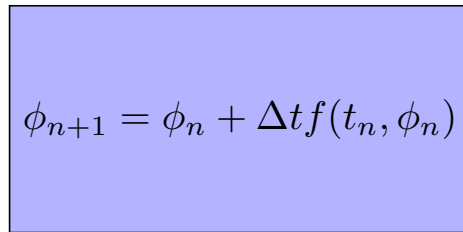
This is the forward Euler method and can be used to solve ODEs of the form:

$$\frac{d\phi}{dt} = f(t, \phi) \quad (5.3)$$

in increments of Δt , by assuming for a small time step the output of the differential is constant. The formula is:

$$\phi_{n+1} = \phi_n + \Delta t f(t_n, \phi_n) \quad (5.4)$$

This formula is used in Figure 5.1 as the ODE solver:



$$\phi_{n+1} = \phi_n + \Delta t f(t_n, \phi_n)$$

Figure 5.2: Forward Euler method solver.

The forward Euler method is easy to start with only the initial conditions being required.

The order of the error can be found by again considering the Taylor series:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) \quad (5.5a)$$

$$f(x+h) = f(x) + hf'(x) + O(h^2) \quad (5.5b)$$

The $O(h^2)$ term is the local truncation error, the error inherited at each time step. Therefore, the local error is proportional to h^2 . The order of the method is however of first

order. I.e the error overall is proportional to h since the number of steps is proportional to $1/h$.

5.3 Modified Euler Method

The problem with the forward Euler method is it makes one prediction about what the solution will be at the next time step and assumes this solution is correct. This method can be modified so that the current and predicted value are used to estimate the gradient over the interval. This gives a more accurate numerical estimation of the gradient so the solution at the next time step should also be more accurate.

Based on this the general formula is:

$$\phi_{n+1} = \phi_n + \frac{\Delta t}{2} \left[f(t_n, \phi_n) + f(t_{n+1}, \tilde{\phi}_{n+1}) \right] \quad (5.6)$$

The term $f(t_{n+1}, \phi_{n+1})$ is the prediction made using the forward Euler method. A flowchart representation of the method is shown in Figure 5.3.

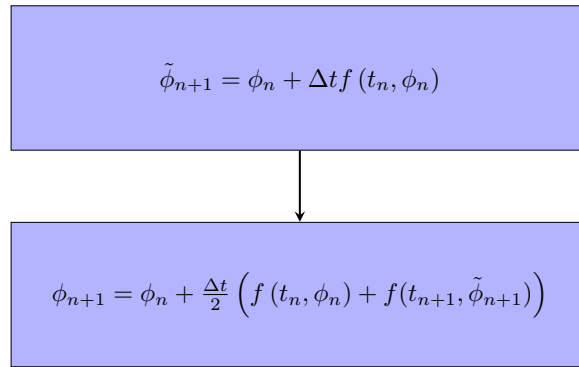


Figure 5.3: Modified Euler method solver.

5.4 Runge-Kutta Method

The modified Euler method (also known as the Heun formula) is a predictor corrector method. In fact it is an example of a more general family of predictor corrector methods known as the Runge-Kutta method. The modified Euler method is in fact a second order Runge-Kutta method:

$$k_1 = \Delta t f(t_n, \phi_n) \quad (5.7a)$$

$$k_2 = \Delta t f(t_n + \Delta t, \phi_n + k_1) \quad (5.7b)$$

$$\phi_{n+1} = y_n + \frac{1}{2}(k_1 + k_2) \quad (5.7c)$$

The predictor corrector process can be extended such that 4 estimates of the solution are made, which are then averaged in a single formula. This gives a fourth order accurate scheme.

$$k_1 = \Delta t f(t_n, \phi_n) \quad (5.8a)$$

$$k_2 = \Delta t f\left(t_n + \frac{\Delta t}{2}, \phi_n + \frac{k_1}{2}\right) \quad (5.8b)$$

$$k_3 = \Delta t f\left(t_n + \frac{\Delta t}{2}, \phi_n + \frac{k_2}{2}\right) \quad (5.8c)$$

$$k_4 = \Delta t f(t_n + \Delta t, \phi_n + k_3) \quad (5.8d)$$

$$\phi_{n+1} = \phi_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (5.8e)$$

More weighting is applied to the terms k_2 and k_3 as these are estimates of the midpoint of the interval. The procedure is shown in Figure 5.4.

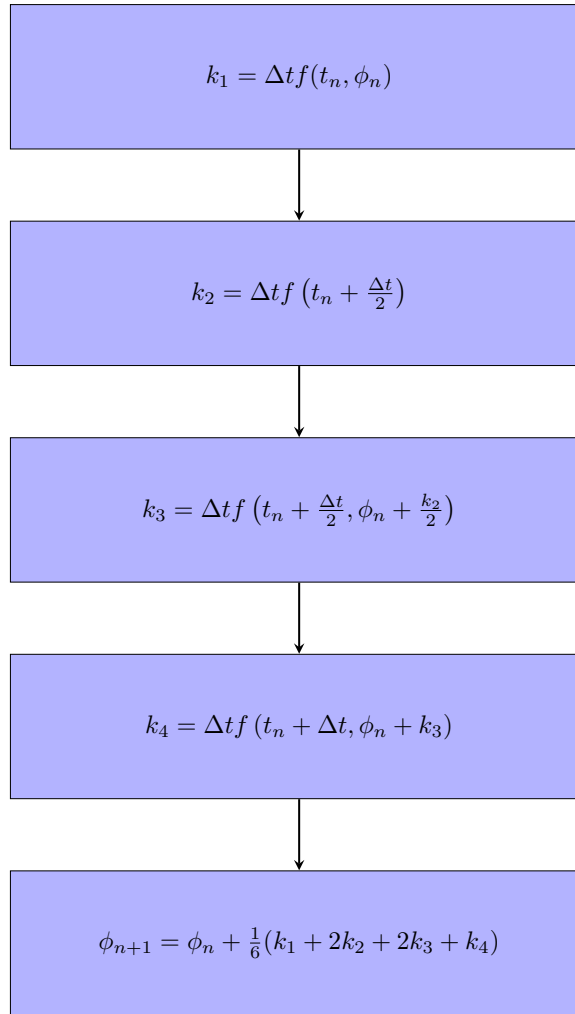


Figure 5.4: Flowchart for the Runge-Kutta method.

5.5 Backward Euler Method

The previously mentioned methods are all examples of explicit methods. So called as a function for the next value in time can be expressed explicitly in terms of known variables. The disadvantage of such methods is they extrapolate into the future; for small time steps this is fine as the error is small. But for larger time steps (relative to the timescale of the problem) this can cause instability and the solution diverges.

An implicit method uses a function with y_{n+1} on both the left and right hand side of the equation. In general this means an iterative method must be used to solve for y_{n+1} . However, for simpler equations an explicit function can be found. The benefit of an implicit method is it is in theory more stable than an explicit method.

A first order accurate implicit method is the backward Euler method:

$$t_{n+1} = t_n + \Delta t \quad (5.9a)$$

$$y_{n+1} = y_n + \Delta t f(t_{n+1}, y_{n+1}) \quad (5.9b)$$

The general solution procedure using this method is shown in Figure x.

Figure x

A non-iterative formula can be derived for the temperature and mass ODES. Unlike the explicit methods, the implicit method will have to be derived for each ODE. For the temperature ODE:

$$\frac{dT_d}{dt} = \frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d} \right) (T_G - T_d) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \quad (5.10)$$

$$t_{n+1} = \Delta t \quad (5.11a)$$

$$T_{d_{n+1}} = T_{d_n} + \Delta t \left[\frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d} \right) (T_G - T_{d_{n+1}}) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right] \quad (5.11b)$$

For the second equation rearrangement is required, the full process is shown in Section B.2:

$$T_{d_{n+1}} = T_{d_n} + \Delta t \left[\frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d} \right) (T_G - T_{d_{n+1}}) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right] \quad (5.12a)$$

$$T_{d_{n+1}} = \frac{T_{d_n} + \Delta t \left[T_G \frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d} \right) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right]}{\left(1 + \Delta t \frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d} \right) \right)} \quad (5.12b)$$

A similar process can be applied to find an equivalent expression for the mass ODE:

$$\frac{dm_d}{dt} = - \frac{Sh}{3Sc_G} \left(\frac{m_d}{\tau_d} \right) H_M \quad (5.13)$$

With:

$$t_{n+1} = \Delta t \quad (5.14a)$$

$$m_{d_{n+1}} = m_{d_n} + \Delta t \left[- \frac{Sh}{3Sc_G} \left(\frac{m_{d_{n+1}}}{\tau_d} \right) H_M \right] \quad (5.14b)$$

Rearranging the second equation leads to:

$$m_{d_{n+1}} = \frac{m_{d_n}}{1 + \Delta t \frac{Sh}{3Sc_G} \left(\frac{H_M}{\tau_d} \right)} \quad (5.15)$$

(The full solution can be found in Section B.3). These numerical methods will be evaluated in the next section to determine their suitability.

6 Single Particle Solution Method

The ODEs for mass and heat transfer are coupled and non-linear making them challenging to solve analytically. To get the simulation running initially the ODEs for heat and mass transfer were solved separately. Different numerical methods were used to solve the ODEs to evaluate their suitability.

6.1 Simulation Settings

For the set of results for the uncoupled heat and mass transfer the following settings were used. The Reynolds number was set to zero, which means the drop is stationary and there is only diffusive heat transfer. This fixes the value of the Nusselt and Sherwood numbers as constant. As M1 is used $f_2 = 1$ and $H_{\Delta T} = 0$.

In addition to this to decouple the temperature ODE from the mass ODE, the term $\left(\frac{L_V}{C_L}\right) \frac{\dot{m}_d}{m_d}$ was ignored. The mass transfer ODE requires no such alterations to decouple it.

Some parameters must be calculated based on empirical relations. For the droplet this includes the μ_V and L_V . See Section x for the formulae.

The full settings are listed in Table x:

6.2 Uncoupled Heat Transfer

Using the settings specified in Section x the temperature ODE becomes:

$$\frac{dT_d}{dt} = \frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d}\right) (T_G - T_d) \quad (6.1)$$

Which can be solved analytically to provide a reference point for the numerical solution. See Section B.4 for the full derivation which produces the solution:

$$T_{d_{n+1}} = T_G - (T_G - T_{d_n}) e^{-\left(\frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d}\right)\right) \Delta t} \quad (6.2)$$

The implementation of equation 6.1 requires the usage of a for loop to iterate through the time steps. Importantly, this loop must check to see if T_{d_n} has exceeded T_G , if this condition is met the loop must stop as the following results will be unphysical.

For plotting data the results have been non-dimensionalised. The points in time are non-dimensionalised with the particle relaxation time constant τ . the temperature values can be non-dimensionalised with the steady state temperature by solving the uncoupled heat transfer equation for T_d when dT/dt is zero:

$$\frac{dT_d}{dt} = \frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d}\right) (T_G - T_d) \quad (6.3a)$$

$$0 = \frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d}\right) (T_G - T_d) \quad (6.3b)$$

$$T_d = T_G \quad (6.3c)$$

The results from a simulation run for one droplet of water evaporating in air for a time step of τ_d are shown in Figure 6.1.

Parameter	Setting
Droplet Properties	
Molecular Weight of Vapour Phase W_V	18.015 <i>kg/kgmol</i>
Boiling Temperature T_B	373.15 <i>K</i>
Latent Heat of Evaporation L_V	$2.257 \times 10^6 + 2.595 \times 10^3(T_B - T_G)$
Density of Liquid Phase ρ_L	997 <i>kg/m³</i>
Specific Heat Capacity of Liquid Phase C_L	4184 <i>J/(kg K)</i>
Initial Temperature T_{d0}	282 <i>K</i>
Initial Diameter squared D_0^2	1.1 <i>mm</i>
Initial Velocity u_0	0 <i>m/s</i>
Initial Reynolds Number Re_0	0
Gas Properties	
Pressure P_G	101325 <i>Pa</i>
Temperature T_G	298 <i>K</i>
Density ρ_G	
Kinematic Viscosity μ_G	
Molecular Weight of Phase W_C	28.97 <i>kg/(kg mol)</i>
Prandtl Number Pr_G	
Schmidt Number Sh_G	
Atmospheric Properties	
Pressure P	$P = P_G$
Temperature T	$T = T_G$
Non-Dimensional Numbers	
Lewis Number	1
Sherwood Number Sh	
Nusselt Number Nu	
Physical Constants	
Gas Constant R	8314.5 <i>J/(K(kg mol))</i>
Universal Gas Constant R	287 <i>J/(kg K)</i>
Specific Heat for Constant Pressure C_{PG}	
Simulation Properties	
Correction factor f_2	1
Driving Potential $H_{\Delta T}$	0
Driving Potential H_M	$\ln(1 + B_M, eq)$

As expected the first order implicit and explicit methods undershoot and overshoot the analytic solution respectively. This is because an explicit method makes an estimate of what the next value will be based on the current gradient. Given the gradient of the analytic solution is always decreasing this means the explicit method is always overestimating the value at the next timestep. A similar argument can be made for an implicit method.

It can also be seen from Figure 6.1 the droplet heats up to the gas temperature within just over $4\tau_h$. As $1/e^4$ is $\approx 2\%$ and the simulation stops once the droplet temperature has reached 99.9% of the gas temperature this proves the droplet temperature increases at the correct rate.

The timestep for example be halved to $\tau/2$ as in Figure 6.2.

Figure 6.2 shows the numerical solutions are closer to the analytic solution compared with Figure 6.1. Therefore demonstrating the numerical solutions converge to the analytic solution as expected.

This convergence is highlighted by Figure 6.3.

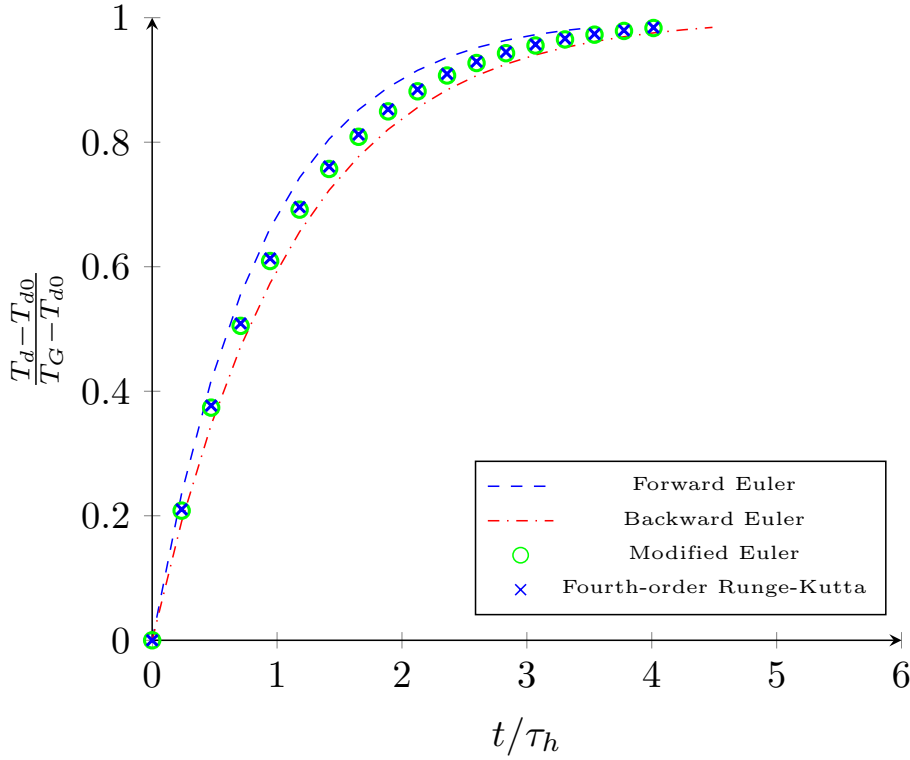


Figure 6.1: T_d for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d0} = 282K$, $T_G = 298K$ and $\Delta t = \tau_{d0}$.

for a timestep very close to zero all the numerical methods appear to have the same average percentage error. This should be zero but is actually 0.3%. This is quite an unusual result, especially when considering the average percentage error decreases with increasing timestep for the modified Euler and Runge-Kutta methods.

This turned out not to be a bug in a code but a bug in the method used for benchmarking the statistics. The simulation stops once $T_d = 0.999T_G$. Removing this stop condition and fixing the simulation lengths using the max number of timesteps produces statistics that make sense:

The simulation duration was fixed to τ_{d0} and the number of timesteps calculated by dividing this by the timestep size. As Figure 6.4 shows all methods converge to zero error with a timestep size of zero. The forward Euler method has perfect linearity with a R^2 value of 0.9998 as calculated by Excel. The backward Euler method has a slightly smaller error, but again has good linearity with $R^2 = 0.9996$. The modified Euler and Runge-Kutta methods now have noticeably different gradients for their error plots. It is just possible to distinguish that the modified Euler method has a quadratic error function, with $R^2 = 0.9996$. Due to how small the average percentage error of the Runge-Kutta method is this has also been plotted separately on Figure 6.5.

The average percentage error as a function of the timestep for the Runge-Kutta method is a quartic as expected given the method is fourth-order. The statistics are again well behaved with $R^2 = 1.000$.

R^2 values close to 1 demonstrate the numerical methods have been implemented correctly as their results are to the correct order. Moreover, as the results for all numerical methods converge to the analytic solution for decreasing timestep size this verifies they are solving the uncoupled temperature case correctly.

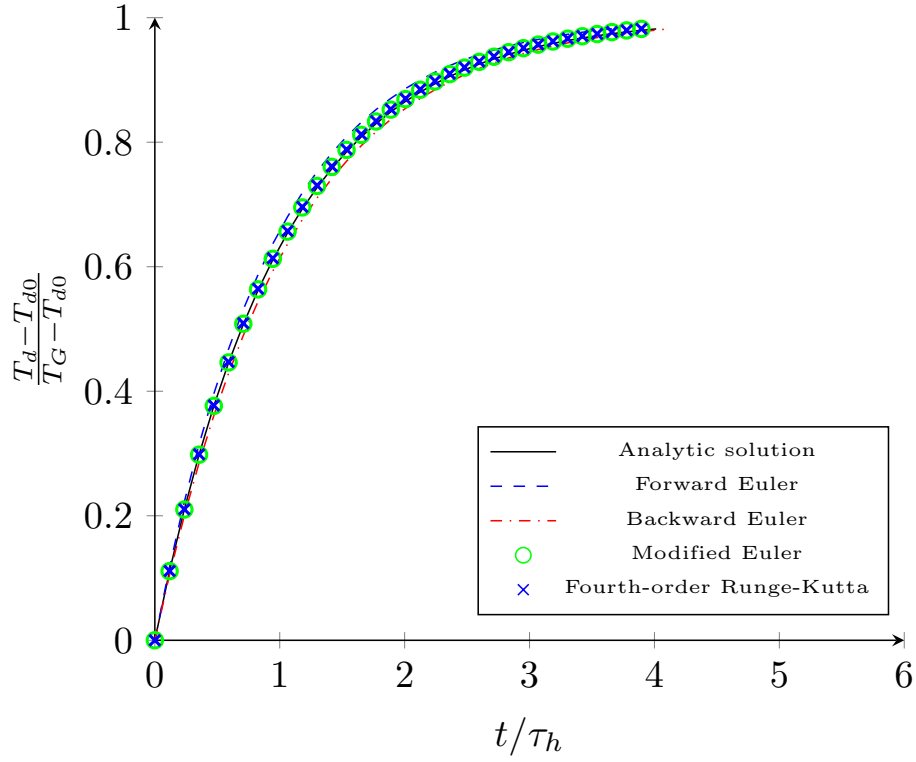


Figure 6.2: T_d for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d0} = 282K$, $T_G = 298K$ and $\Delta t = \tau/2$.

6.3 Uncoupled Mass Transfer

The mass transfer equation is:

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} \frac{m_d}{\tau_d} H_M \quad (6.4)$$

The literature commonly plots D^2 instead of mass, solving equation 6.4 for diameter, requires a relation between diameter D and m_d :

$$m_d = \rho_d V_d \quad (6.5a)$$

$$m_d = \rho_d \left(\frac{4}{3}\right) \pi \left(\frac{D}{2}\right)^3 \quad (6.5b)$$

$$m_d = \frac{\rho_d \pi D^3}{6} \quad (6.5c)$$

So:

$$\frac{dm_d}{dt} = \frac{\rho_d \pi D^2}{2} \frac{dD}{dt} \quad (6.6)$$

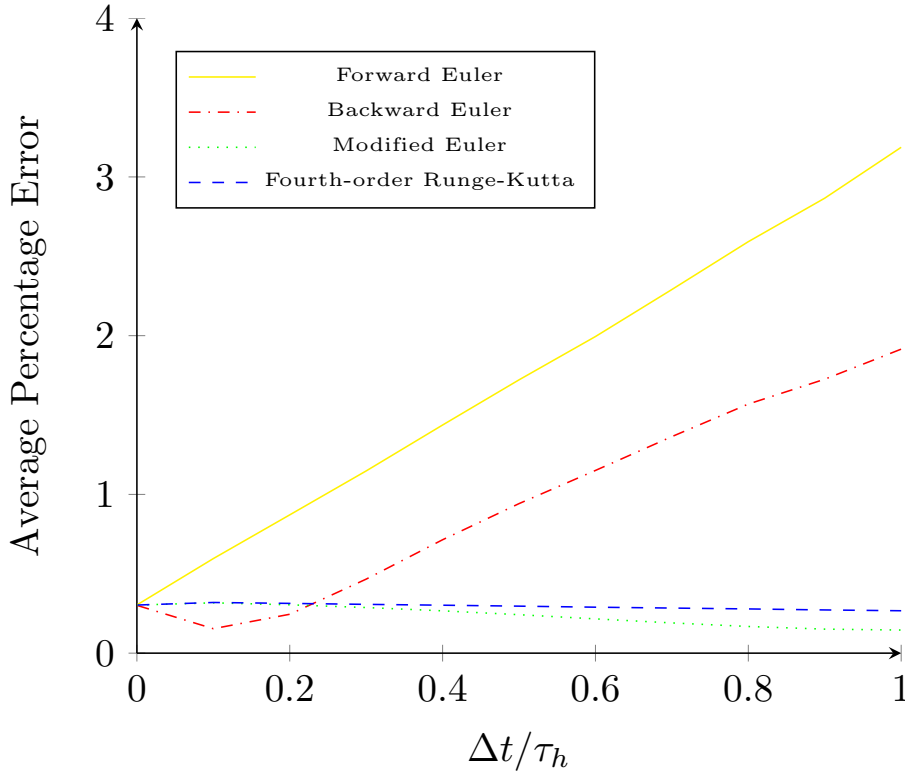


Figure 6.3: Average percentage error compared with the analytic solution for a range of timesteps and numerical methods.

These can be substituted into equation 6.4, along with $\tau_d = \rho_d D^2 / (18\mu_g)$:

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} \frac{m_d}{\tau_d} H_M \quad (6.7a)$$

$$\frac{\rho_d \pi D^2}{2} \frac{dD}{dt} = -\frac{Sh}{3Sc_G} \frac{18\rho_d \pi D^3 \mu_g}{6\rho_d D^2} H_M \quad (6.7b)$$

$$\frac{1}{2} [D_{n+1}^2 - D_n^2] = -\frac{2Sh}{Sc_G \rho_d} \mu_g H_M \Delta t \quad (6.7c)$$

$$D_{n+1}^2 = D_n^2 - \frac{4Sh}{Sc_G \rho_d} \mu_g H_M \Delta t \quad (6.7d)$$

A numerical solution to the D^2 ODE can be found as:

$$D_{n+1} = D_n - \Delta t \left(\frac{2Sh}{Sc_G \rho_d} \mu_g H_M \frac{1}{D_n} \right) \quad (6.8)$$

A rearrangement of equation 6.5 gives the droplet diameter as:

$$m_d = \frac{\rho_d \pi D^3}{6} \quad (6.9a)$$

$$D = \left(\frac{6}{\rho_d \pi} \right)^{1/3} (m_d)^{1/3} \quad (6.9b)$$

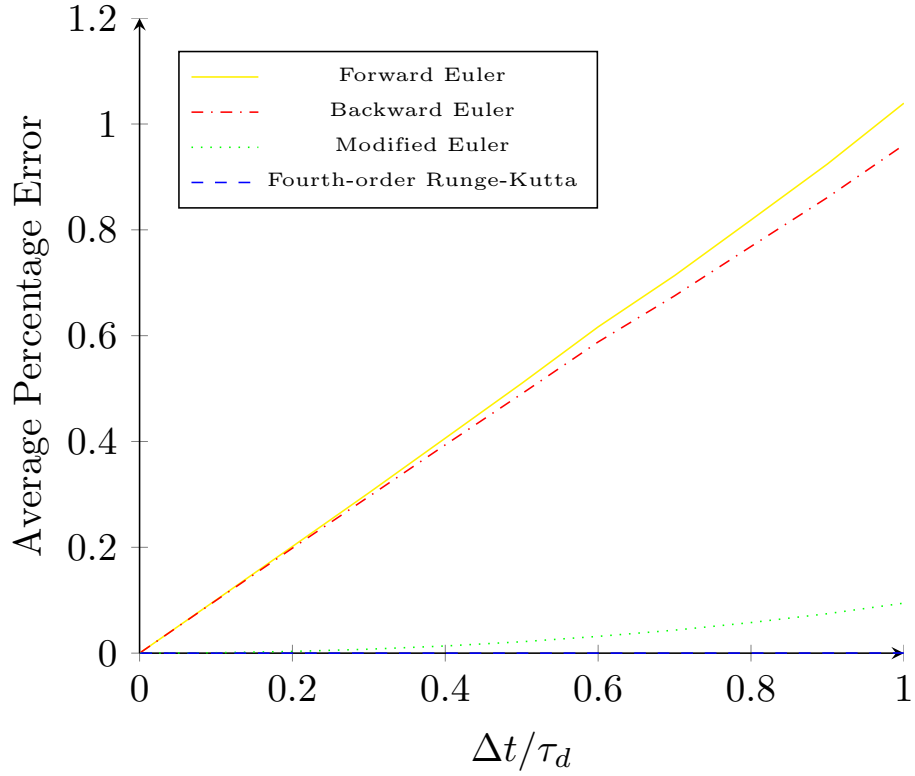


Figure 6.4: Average percentage error compared with the analytic solution for a range of timesteps and numerical methods.

Therefore:

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} \frac{m_d}{\tau_d} H_M \quad (6.10a)$$

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} H_M m_d \left(\frac{18\mu_G}{\rho_d D^2} \right) \quad (6.10b)$$

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} H_M m_d \left(\frac{18\mu_G}{\rho_d} \right) \left(\left(\frac{6}{\rho_d \pi} \right)^{-2/3} (m_d)^{-2/3} \right) \quad (6.10c)$$

$$(m_{dn+1})^{2/3} = (m_{dn})^{2/3} - \frac{2Sh}{3Sc_G} \mu_G H_M \left(\frac{6}{\rho_d} \right)^{1/3} \pi^{2/3} \Delta t \quad (6.10d)$$

The full integration steps for the mass and diameter equations are given in Section B.5.

Calculating H_M :

$$H_M = \ln [1 + B_{M,eq}] \quad (6.11)$$

Where $B_{M,eq}$ is the Spalding transfer number for mass transfer given by:

$$B_{M,eq} = \frac{Y_{s,eq} - Y_G}{1 - Y_{s,eq}} \quad (6.12)$$

With Y_G the free stream vapour mass fraction and $Y_{s,eq}$ the vapour mass fraction at the droplet's surface:

$$Y_{s,eq} = \frac{\chi_{s,eq}}{\chi_{s,eq} + (1 - \chi_{s,eq})\theta_2} \quad (6.13)$$

$\chi_{s,eq}$ is the surface equilibrium mole fraction of the vapour and θ_2 is the ratio of molecular weights:

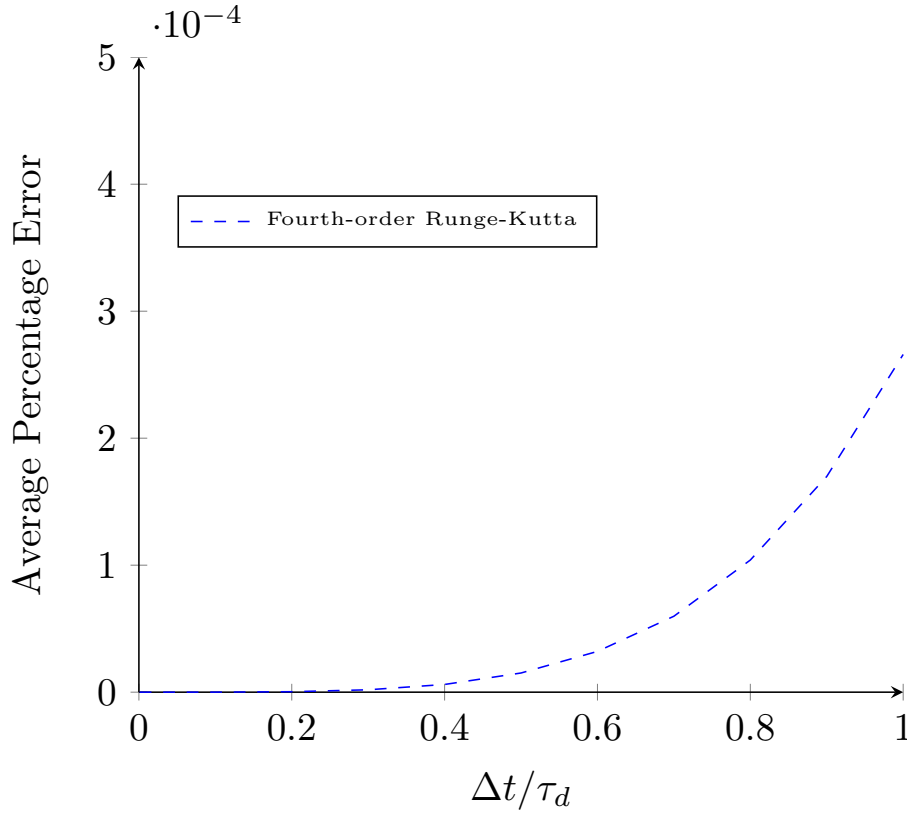


Figure 6.5: Average percentage error compared with the analytic solution for the Runge-Kutta method for a range of timesteps.

$$\theta_2 = \frac{W_C}{W_V} \quad (6.14)$$

For water evaporating in air $W_V = 18.015 \text{ kg}/(\text{kg mole})$ is the molecular weight of water vapour and $W_C = 28.97 \text{ kg}/(\text{kg mole})$ is the molecular weight of air. So $\theta_2 = 1.61$.

The Clausius-Clapeyron equation for constant latent heat gives a relation between the saturation pressure P_{sat} and the free stream pressure P_G as:

$$\chi_{s,eq} = \frac{P_{sat}}{P_G} \quad (6.15a)$$

$$\chi_{s,eq} = \frac{P_{atm}}{P_G} \exp \left[\frac{L_V}{\bar{R}/W_V} \left(\frac{1}{T_B} - \frac{1}{T_d} \right) \right] \quad (6.15b)$$

Where:

- $P_{atm} = 101325 \text{ Pa}$
- $P_G = \rho_G R T_G = (1.225 \text{ kg}/\text{m}^3)(287 \text{ J}/\text{kg K})(298 \text{ K}) = 104769 \text{ Pa}$
- $\bar{R} = 8.3145 \text{ J}/(\text{K mole}) = 8314.5 \text{ J}/(\text{K kg mol})$
- $L_V = 2.257 \times 10^6 + 2.595 \times 10^3(373.15 - T) \text{ J}/\text{kg}$ Latent heat of evaporation for water
- $T_B = 373.15 \text{ K}$ boiling temperature of water

The conditions used:

- Droplet species - water
- Gas species - air
- $m_{d,0} = 0.001 \text{ kg}$
- $D_{d,0} = 1.24 \text{ mm}$
- $T_G = 365 \text{ K}$
- $T_{d,0} = 350 \text{ K}$
- $Re_d = 0$
- $L_V = 2.45 \times 10^6 \text{ J/kg}$ using $T = T_G$

This gives $\chi_{s,eq}$ as:

$$\chi_{s,eq} = \frac{101325 \text{ Pa}}{104769 \text{ Pa}} \exp \left[\frac{2.45 \times 10^6 \text{ J/kg}}{8314.5 \text{ J/(K kg mol)}/18.015 \text{ kg/(kg mole)}} \left(\frac{1}{373.15 \text{ K}} - \frac{1}{365 \text{ K}} \right) \right] \quad (6.16a)$$

$$\chi_{s,eq} = 0.967 \exp \left[(5308 \text{ K})(-5.98 \times 10^{-5} / \text{K}) \right] \quad (6.16b)$$

$$\chi_{s,eq} = 0.967 \exp[-0.318] \quad (6.16c)$$

$$\chi_{s,eq} = 0.704 \quad (6.16d)$$

Therefore $Y_{s,eq}$ is:

$$Y_{s,eq} = \frac{\chi_{s,eq}}{\chi_{s,eq} + (1 - \chi_{s,eq})\theta_2} \quad (6.17a)$$

$$Y_{s,eq} = \frac{0.704}{0.704 + (1 - 0.704)1.61} \quad (6.17b)$$

$$Y_{s,eq} = 0.596 \quad (6.17c)$$

And:

$$B_{M,eq} = \frac{Y_{s,eq} - Y_G}{1 - Y_{s,eq}} \quad (6.18a)$$

$$B_{M,eq} = \frac{0.596 - 0}{1 - 0.596} \quad (6.18b)$$

$$B_{M,eq} = 1.48 \quad (6.18c)$$

Finally:

$$H_M = \ln [1 + B_{M,eq}] \quad (6.19a)$$

$$H_M = \ln [1 + 1.48] \quad (6.19b)$$

$$H_M = 0.908 \quad (6.19c)$$

The mass transfer solution has been plotted in Figure 6.6 for $\Delta t = \tau_{au_d}$. For the numerical solution τ_d was recalculated at the start of each timestep. At the end of each timestep the mass value is used to calculate D^2 , which is non-dimensionalised with the start diameter.

The solution from the modified Euler and Runge-Kutta methods have been plotted separately on Figure 6.7.

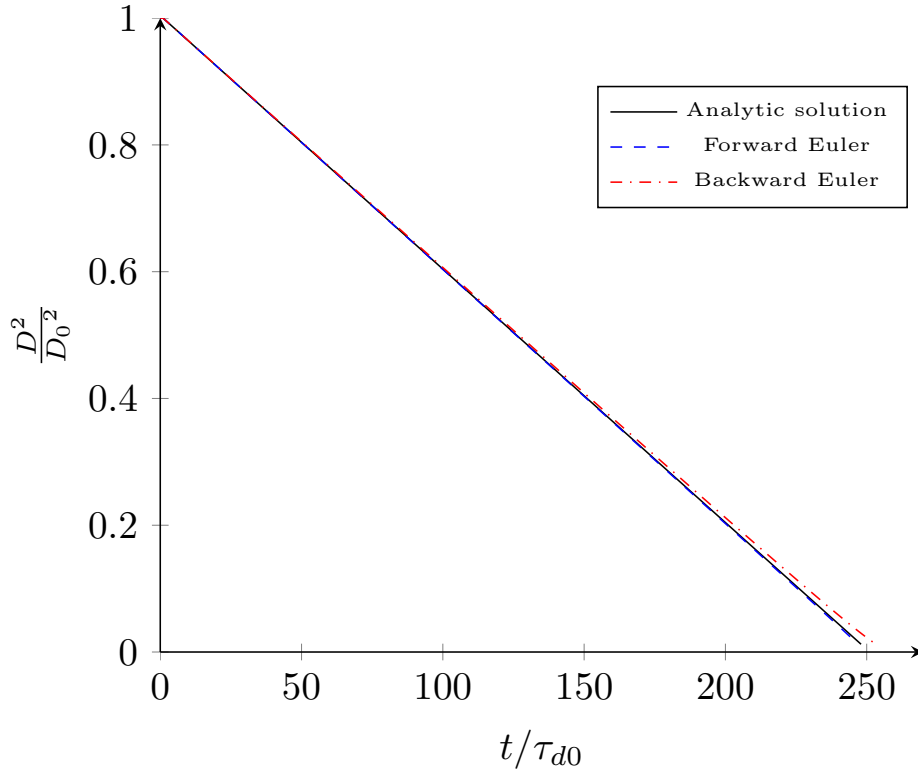


Figure 6.6: Non-dimensionalised D^2 for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d_0} = 282$, $T_G = 298$, $Y_G = 0$ and $\Delta t = \tau_d$.

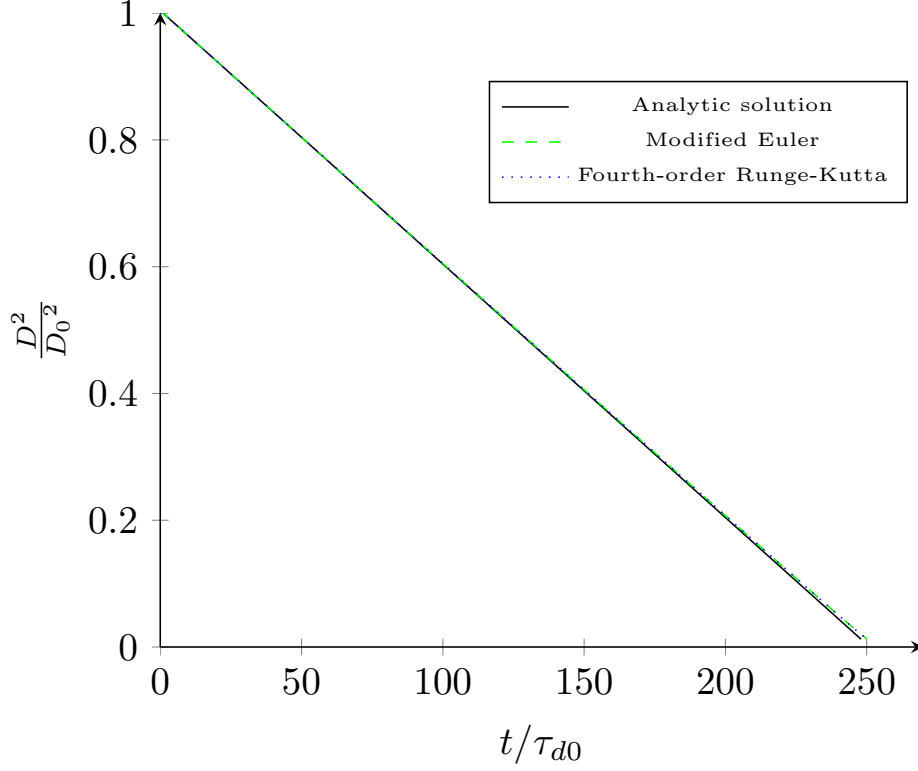


Figure 6.7: Non-dimensionalised D^2 for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d_0} = 282$, $T_G = 298$, $Y_G = 0$ and $\Delta t = \tau_d$.

Much like was the case for the temperature solution the explicit methods overshoot the

analytic solution. The exception being the forward Euler method under-predicts the analytic solution and appears to better predict the the analytic solution. The droplet evaporation time has been non-dimensionalised with the initial droplet momentum timescale as this changes as the droplet's mass decreases.

Note that unlike the heat transfer solution the results of the mass transfer plotted as diameter squared give a linear relation. This is intrinsic to the assumption the droplet temperature remains at the wet bulb temperature. Therefore the rate of evaporation is only dependent on surface area which is a function of D^2 .

To confirm the numerical methods converge to the analytic solution a similar convergence plot to Figure 6.4 has been plotted in Figure 6.8. For this convergence test the simulation time was limited by the mass stop condition.

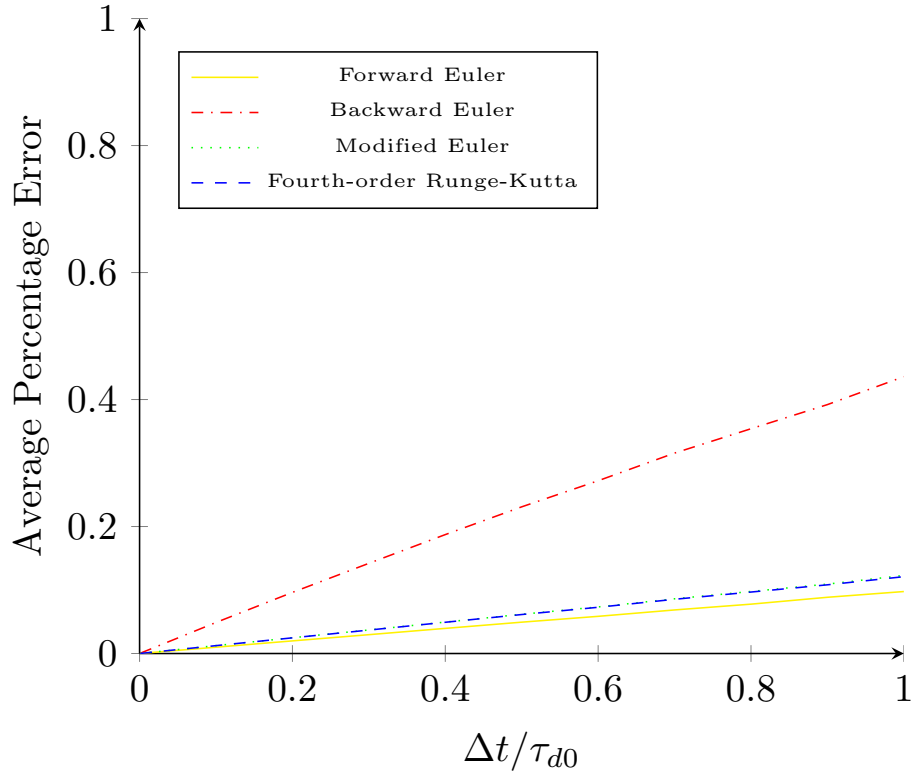


Figure 6.8: Average percentage error compared with the analytic solution for a range of numerical methods and timesteps.

As should be the case all the numerical methods converge to zero percentage error for a timestep of zero. What is unusual is the modified Euler and Runge-Kutta methods have a slightly larger error than the forward Euler method. This may be because the high order methods are multistep. At each step within the method a prediction about what the solution is, is made. For all steps the same τ_d is used. This would explain why the modified Euler and Runge-Kutta methods have such a similar error function.

This can be fixed by updating τ_d for each step which as can be seen in Figure 6.9 and 6.10 the error functions are now as expected.

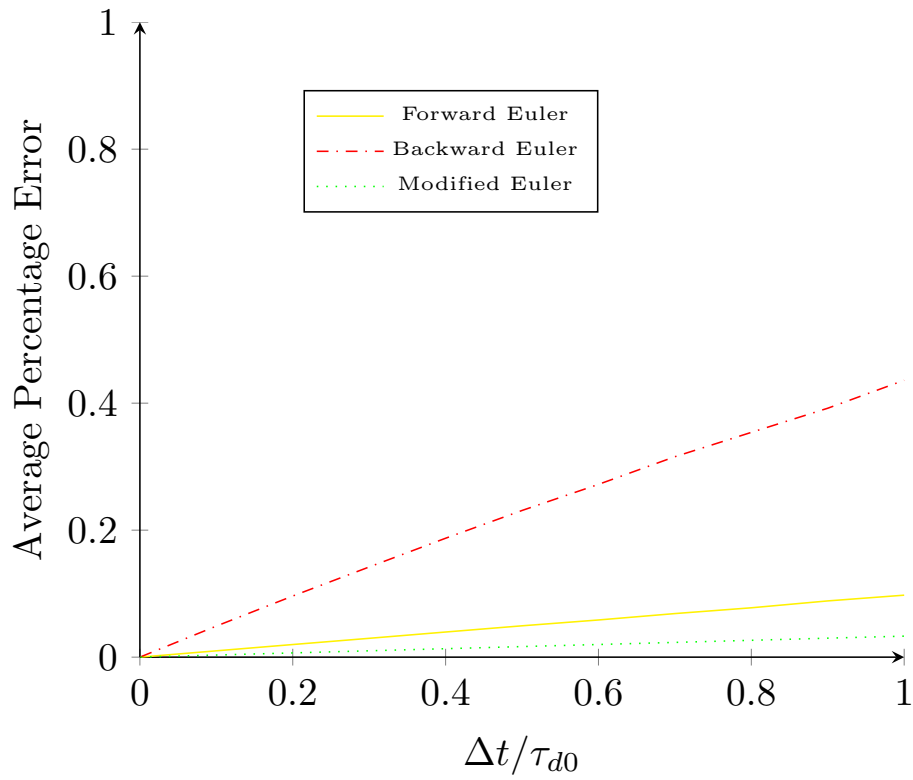


Figure 6.9: Average percentage error compared with the analytic solution for a range of numerical methods and timesteps.

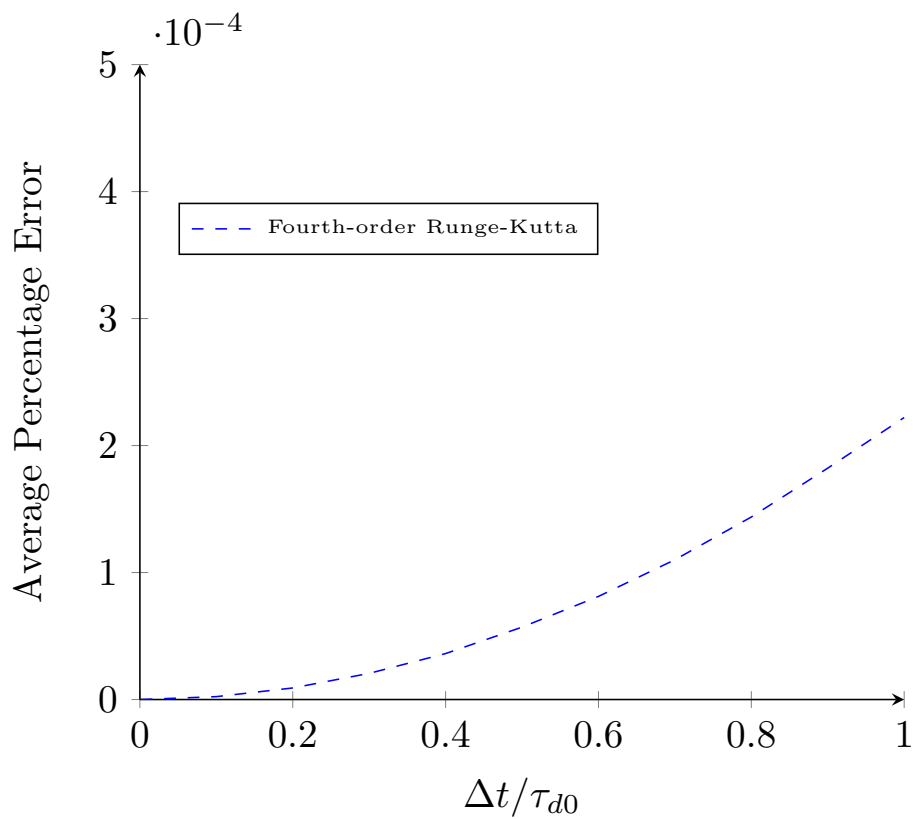


Figure 6.10: Average percentage error compared with the analytic solution for the Runge-Kutta method and a range of timesteps.

Like with the temperature convergence the R^2 values for the D^2 error functions can be calculated. They are as follows:

- Forward Euler - $R^2 = 0.9976$
- Backward Euler - $R^2 = 0.9999$
- Modified Euler - $R^2 = 1.000$
- Runge-Kutta - $R^2 = 1.000$

This affirms the numerical methods perform as expected.

The verification tests here confirm the importance of verifying simulation code to find bugs.

6.4 Coupled Heat and Mass Transfer

Solving the heat and mass transfer ODEs coupled is not a trivial task due to their non-linear nature. An analytic solution for the ODEs coupled has not been found so the equations can only be solved numerically.

The following figures use the data from Miller et al. corresponding to Figure 2 in that paper. The testing procedure is the same as for the uncoupled cases. With multiple numerical methods and timestep sizes being tested.

Figures ?? and ?? show the results for a forward Euler method and Figures ?? and ?? show the results for a fourth order Runge-Kutta method. For both methods a timestep of $0.01s$ was used

Figures ?? and ?? shows the effect of increasing the timestep Δt .

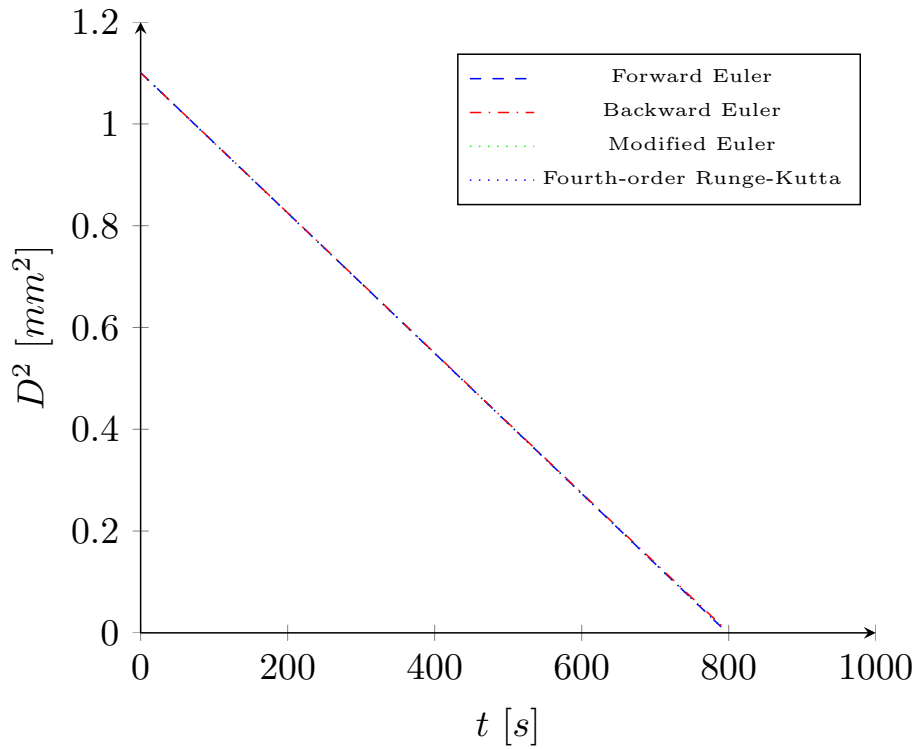


Figure 6.11: D^2 for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d_0} = 282K$, $T_G = 298K$, $Y_G = 0$ and $\Delta t = \tau_d/8$.

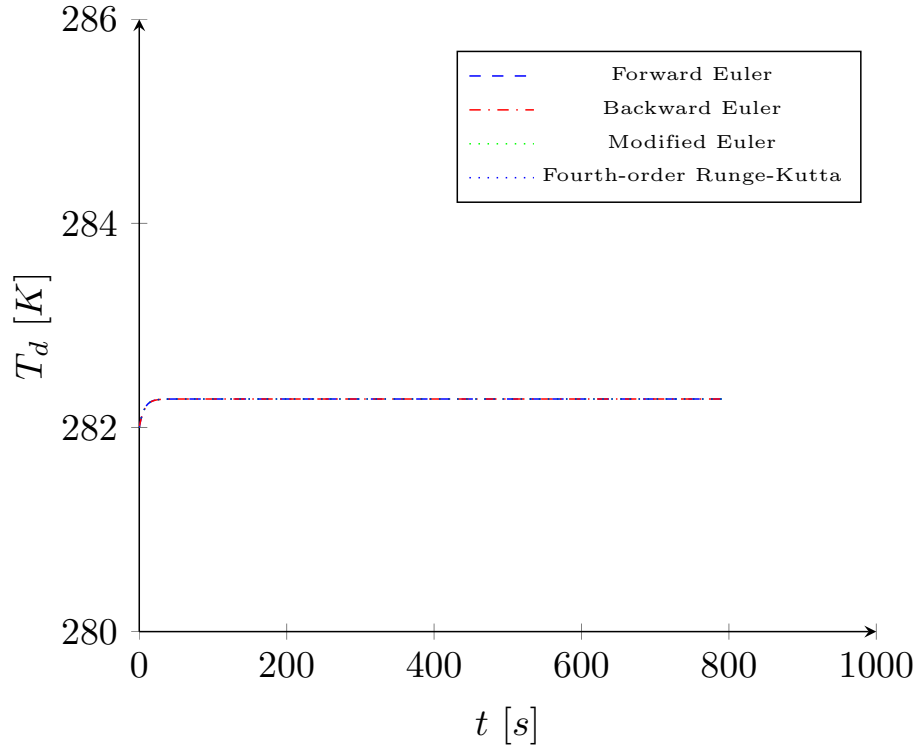


Figure 6.12: T_d for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d0} = 282K$, $T_G = 298K$, $Y_G = 0$ and $\Delta t = \tau_d/8$.

The droplet quickly reaches an equilibrium temperature of $282.27K$ and remains at this temperature as the droplet evaporates at a steady rate. There is little variation between the numerical methods, but this becomes more evident if the timestep is reduced. Using a timestep of $\tau_d/2$ it can be seen the explicit methods diverge quite significantly at the end of the simulation in Figures 6.13 and 6.14.

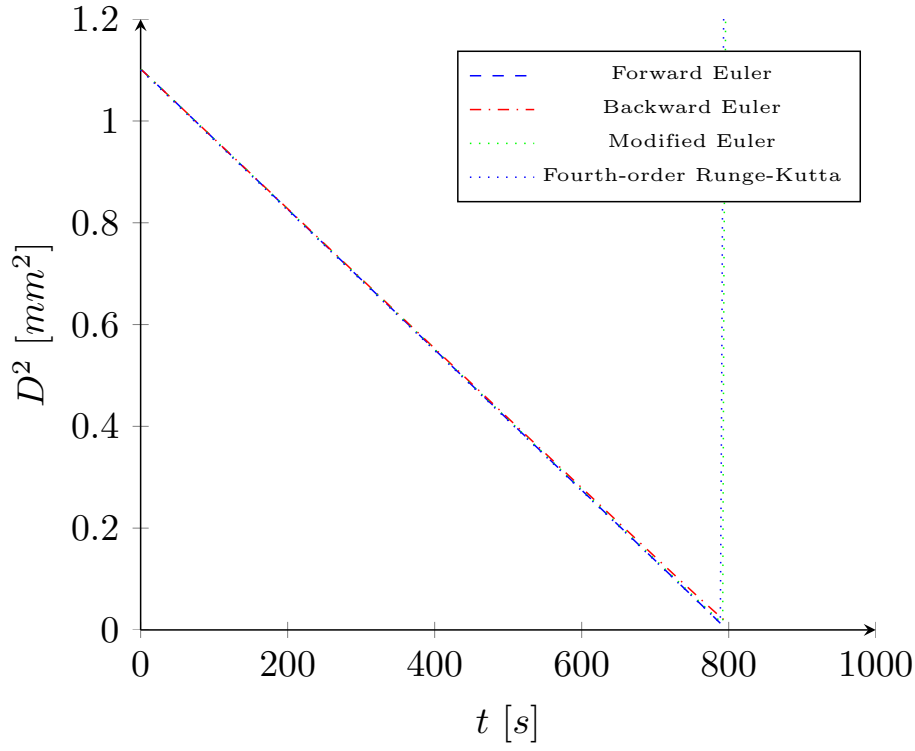


Figure 6.13: D^2 for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d0} = 282K$, $T_G = 298K$, $Y_G = 0$ and $\Delta t = \tau_d/2$.

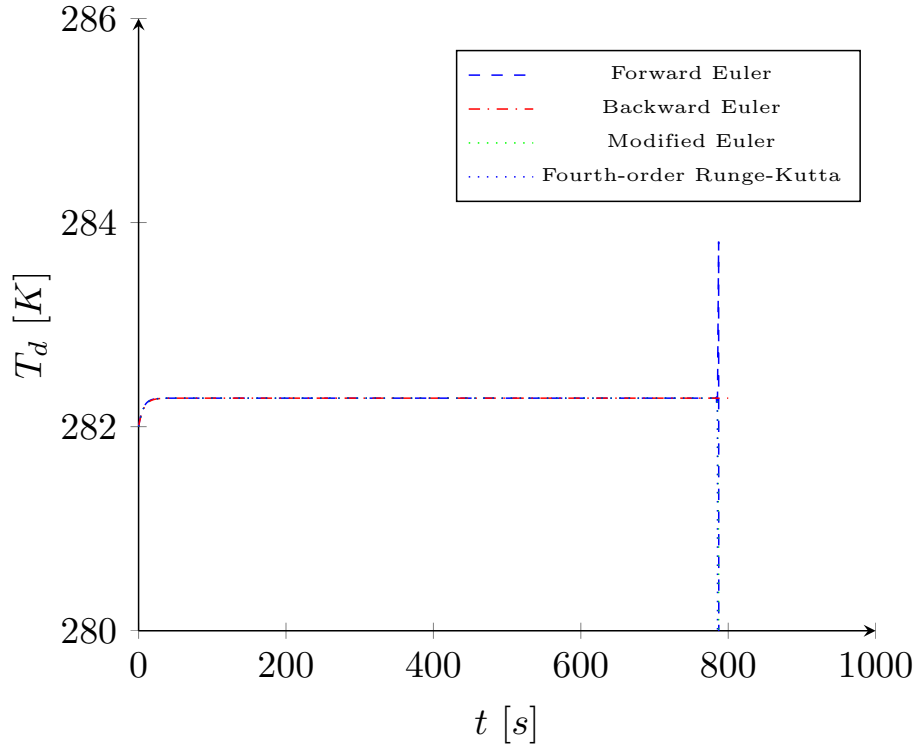


Figure 6.14: T_d for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d0} = 282K$, $T_G = 298K$, $Y_G = 0$ and $\Delta t = \tau_d/2$.

Plotting the results non-dimensionally using τ_d makes it clear what causes this:

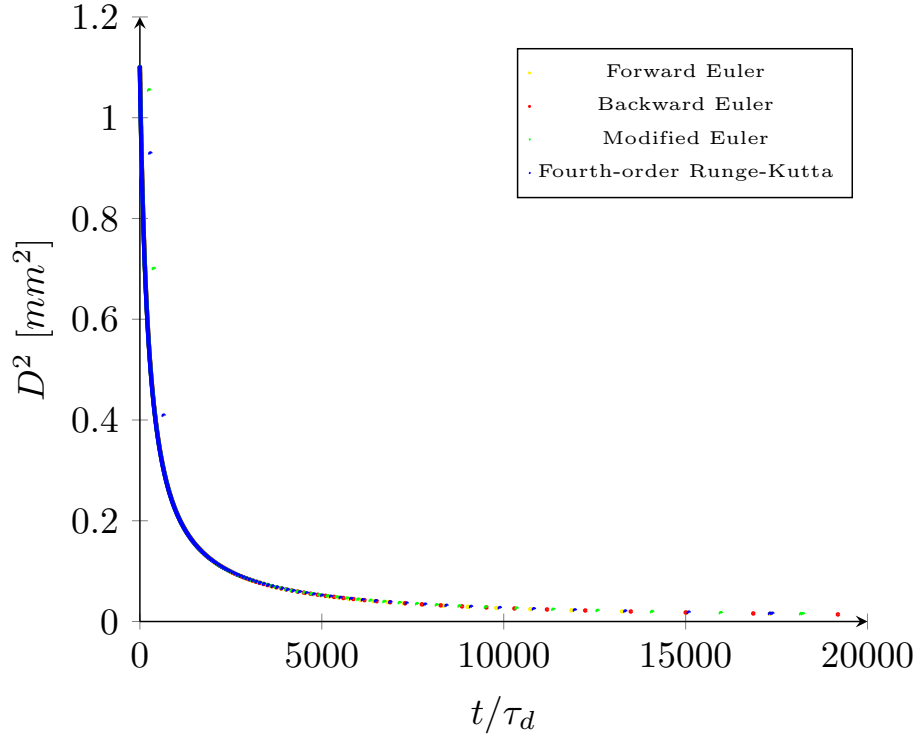


Figure 6.15: D^2 for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d_0} = 282K$, $T_G = 298K$, $Y_G = 0$ and $\Delta t = \tau_d/2$.

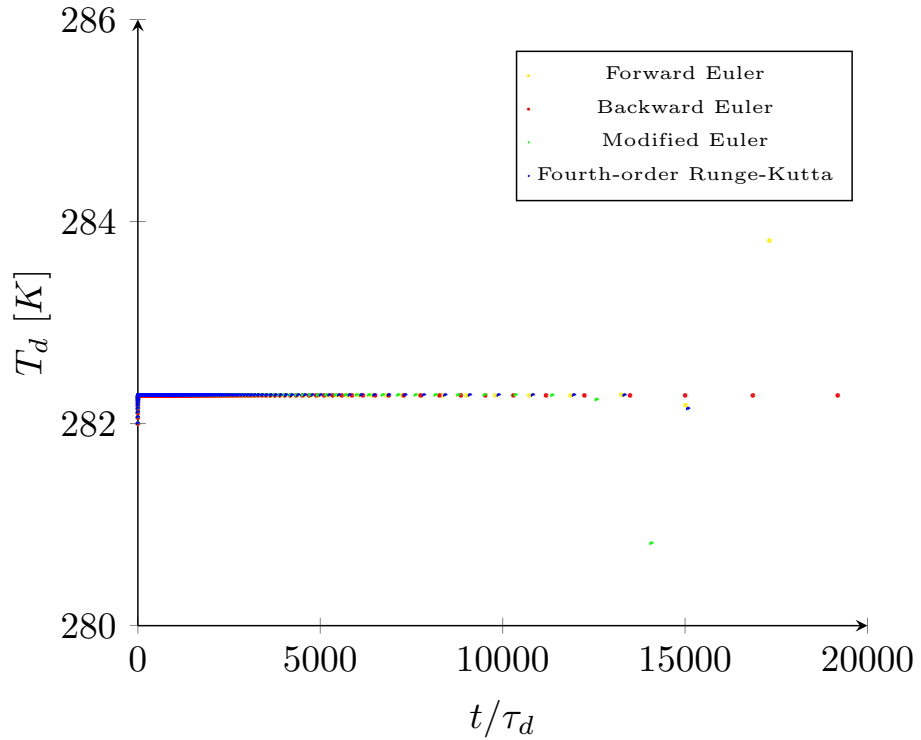


Figure 6.16: T_d for a droplet sized $D^2 = 1.1mm$ with $Re_d = 0$, $T_{d_0} = 282K$, $T_G = 298K$, $Y_G = 0$ and $\Delta t = \tau_d/2$.

The momentum timescale of the droplet, τ_d decreases as the droplet diameter decreases. This means the timescale of the physics increases relative to the size of the timestep as the ODEs are a function of $1/\tau_d$. As explicit methods extrapolate based on the current value

to find the next value; the error incurred in the extrapolation increases with the size of timestep. A solution to this problem is dynamic timestepping which involves changing the size of the timestep as a function of τ_d . For one droplet this procedure is simple enough, but for multiple droplets the timestep is common across all droplets. This means basing the timestep size on the smallest droplet timescale, which could be very inefficient. Consider a scenario where one droplet at the start of the simulation a droplet rapidly evaporates, this means a small timestep size for droplets which may be very slowly evaporating.

Therefore, an appropriate solution is to use an implicit method. The backward Euler method is appropriate to use here given it does not suffer the same instability as the explicit methods. Further, it permits the usage of larger timesteps which is advantageous in reducing computation time. Other higher order implicit methods are available, but at the cost of an iterative procedure at each timestep to solve the equations. Therefore the implicit Euler method will be used for the OpenCL simulation code.

7 Verification and Validation of the Existing Code

Before the heat and mass transfer methods can be added to the existing code some modifications to it had to be made. The existing code calculates the position and velocity of the particles but also includes a discrete element method (DEM) to simulate the collision of particles. The latter is computationally expensive and is not needed for the heat and mass transfer calculations. The DEM code can be “switched off” so the existing code solves just for the particle position and velocity. There are some checks that need to be done to ensure the modified code is solving the equations correctly and producing physical results.

7.1 Analytic Test Case

One very simple check that can be made initially is to check if results from the code converge to some analytic solution. The ideal case for this is the droplet accelerating under gravity to reach a terminal velocity. The differential equation to be solved is:

$$\frac{du}{dt} = g + \frac{1}{\tau_d}(u_f - u) \quad (7.1)$$

The full solution procedure can be found in Section B.1. This leads to:

$$u = \tau_d \left(g + \frac{u_f}{\tau_d} \right) (1 - e^{-t/\tau_d}) \quad (7.2)$$

To simplify this further the fluid velocity can be set to zero, yielding the simplified formula:

$$u = \tau_d g (1 - e^{-t/\tau_d}) \quad (7.3)$$

The settings used for the test case are shown in Table x:

Parameter	Setting
Droplet Properties	
Density of Liquid Phase ρ_L	997 kgm^{-3}
Initial Diameter D_0	0.05 m
Initial Velocity u_0	0 ms^{-1}
Gas Properties	
Kinematic Viscosity μ_G	$0.1 \text{ kgm}^{-1} \text{ s}^{-1}$
Physical Constants	
Gravity g	9.81 ms^{-1}

Table 7.1: Simulation settings used for acceleration under gravity test case.

For non-dimensionalising the results τ_d has been used for time and the terminal velocity has been used u_i . See Section x for a derivation of the terminal velocity formula shown in equation x.

$$v_{final} = u - g \frac{\rho_d D^2}{18\mu_G} \quad (7.4)$$

Using the settings from Table x the terminal velocity is:

$$v_{final} = 0 \text{ ms}^{-1} - (9.81 \text{ ms}^{-2}) \left(\frac{(997 \text{ kgm}^{-3})(0.05 \text{ m})^2}{(18)(0.1 \text{ kgm}^{-1} \text{ s}^{-1})} \right) \quad (7.5a)$$

$$v_{final} = -13.58 \text{ ms}^{-1} \quad (7.5b)$$

Figure 7.1 shows the analytic and numerical solutions for the gravity test case. This shows the particle accelerating to the terminal velocity. As explained earlier as the numerical solution for the velocity uses an implicit method so this under predicts the analytic solution.

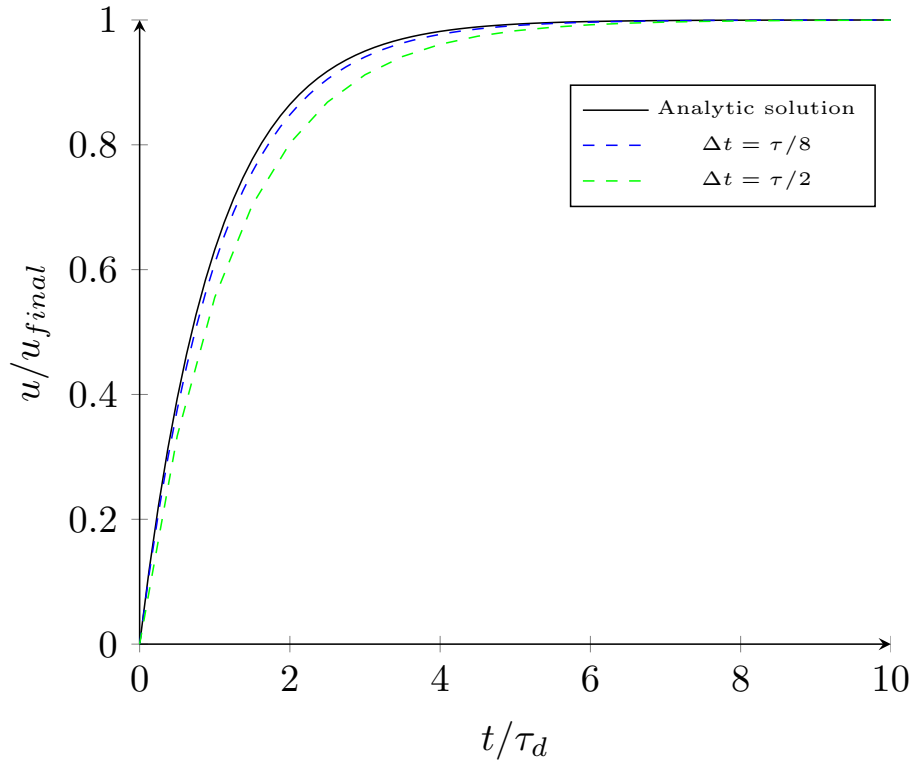


Figure 7.1: Non-dimensionalised velocity for a droplet sized $D^2 = 50mm$ with $Re_{d0} = 0$, and $\Delta t = \tau_d/2$ and $\Delta t = \tau_d/8$.

As with previous tests it is important to check the solution produced by the code converges to the analytic solution. For this test case the simulation time was set as 21 seconds which is $\sim 15\tau_d$. The results from this are shown in Figure 7.2.

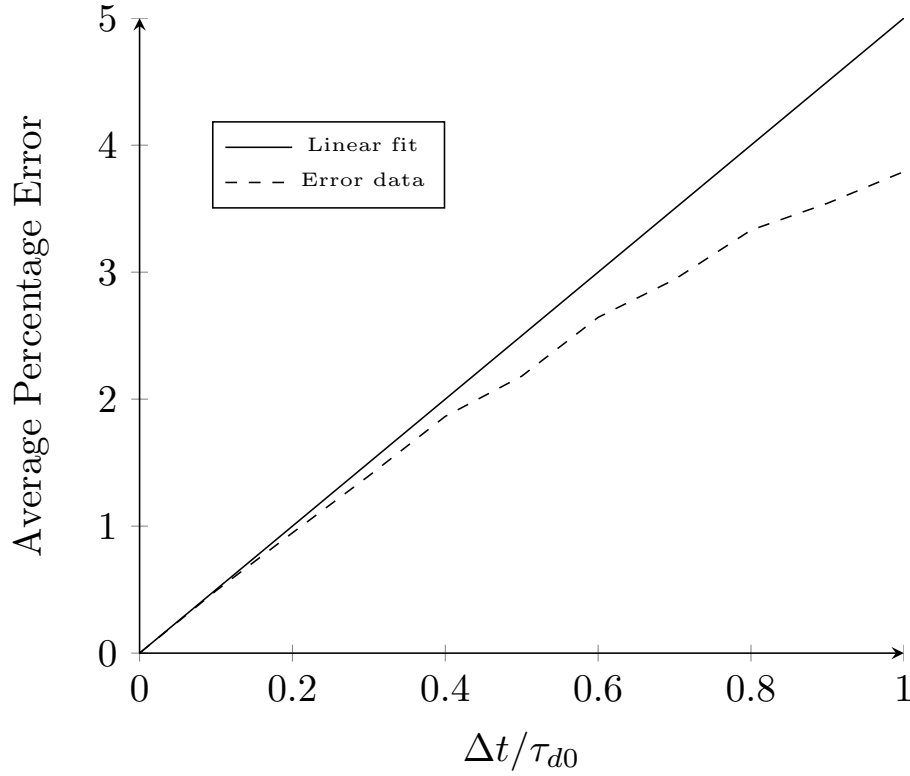


Figure 7.2: convergence of numerical results for gravity test case for a droplet sized $D^2 = 50mm$ with $Re_{d0} = 0$, and a range of timesteps.

Figure 7.2 clearly indicates convergence for the gravity test case. As well as confirming the velocity equation is solved with higher than first order accuracy as reported in [1].

7.2 Statistical Test Case

Another check that can be made is to find if the code produces statistically stationary conditions. This can be done by calculating the relative velocity. I.e. the difference between the particle velocity and the fluid velocity at different times. The particle velocity is a Lagrangian variable (only varying with time). Whilst the fluid velocity is an Eulerian variable as it varies with both space and time. To calculate the relative velocity between the particle and fluid a Lagrangian description is needed for velocity.

The averaging procedure to get the relative velocities is:

$$\bar{U}_{x,rel}(t) = \sum_{i=1}^N \frac{[U_{p_i}(t) - U_{f_i}(t)]}{N} \quad (7.6a)$$

$$\bar{V}_{y,rel}(t) = \sum_{i=1}^N \frac{[V_{p_i}(t) - V_{f_i}(t)]}{N} \quad (7.6b)$$

$$\bar{W}_{z,rel}(t) = \sum_{i=1}^N \frac{[W_{p_i}(t) - W_{f_i}(t)]}{N} \quad (7.6c)$$

Where for each velocity component the relative velocity is found for all particles and averaged to produce the mean velocity component. The behaviour of the mean relative velocity will be random initially but should converge to some value at a later point in time. This behaviour should be the same for the other statistical moments.

The Stokes number will dictate the behaviour of the particles. If it is $\ll 1$ the particles are massless and should follow the flow. If the Stokes number is $\gg 1$ the particles are

essentially billiard balls and the flow field of the velocity has no bearing on the motion of the particles.

The conditions used for the simulation of a Taylor-Green vortex were:

Parameter	Setting
Droplet Properties	
Density of Liquid Phase ρ_L	2000 kg/m^3
Initial Diameter D_0	0.05 m
Gas Properties	
Kinematic Viscosity μ_G	see Table 7.3
Flow Magnitude A	5
Vortex Frequency a	3
Non-Dimensional Numbers	
Stokes Number St	see Table 7.3
Physical Constants	
Gravity g	0 ms^{-1}
Simulation Properties	
Number of Particles	100

Table 7.2: Simulation settings used for statistical test case.

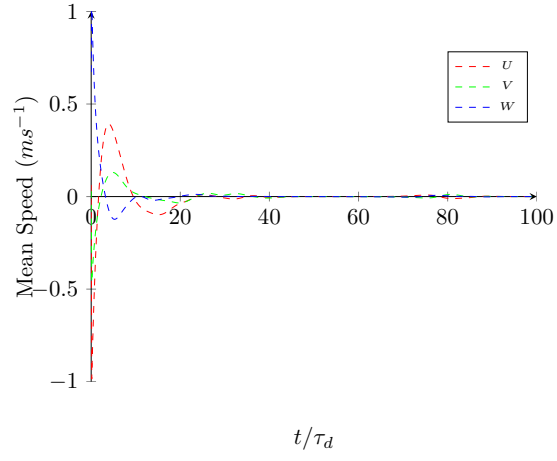
Kinematic Viscosity	Stokes Number
10.40	0.1
1.04	1.0
0.1040	10.0

Table 7.3: Settings used to get different Stokes numbers.

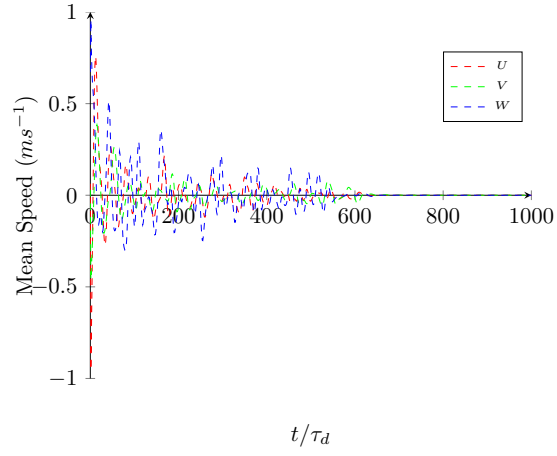
The distribution of starting conditions gives a mean particle speed of $1ms^{-1}$ and standard deviation of $0.1ms^{-1}$. With the particle directions evenly distributed.

Figure 7.3 shows the change in relative velocity between the particles and the fluid. As the Stokes number is increased the particles take longer to respond to the fluid. For a small Stokes number it can be seen that within around 20 momentum relaxation time constants the particle velocity has converged to the fluid velocity. This is further shown by Figure 7.4a where the angle between the particle and fluid velocity converges to zero.

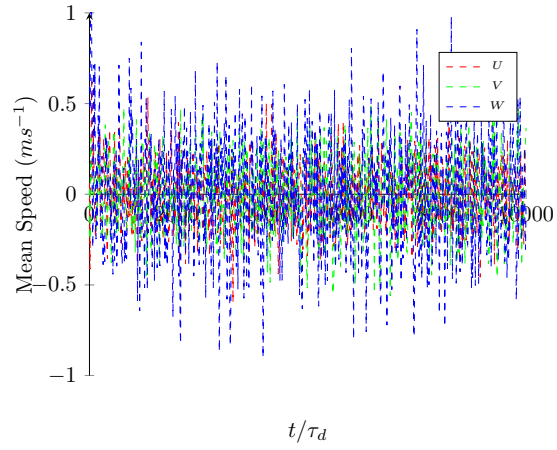
A similar behaviour is present for a Stokes number of 1. However, it takes longer for the relative speeds to converge to zero. By contrast for a Stokes number of 10 the relative speeds and relative angles do not converge at all. This is to be expected as the fluid should have little bearing on the motion of the particles for this Stokes number.



(a) Stokes number of 0.1.

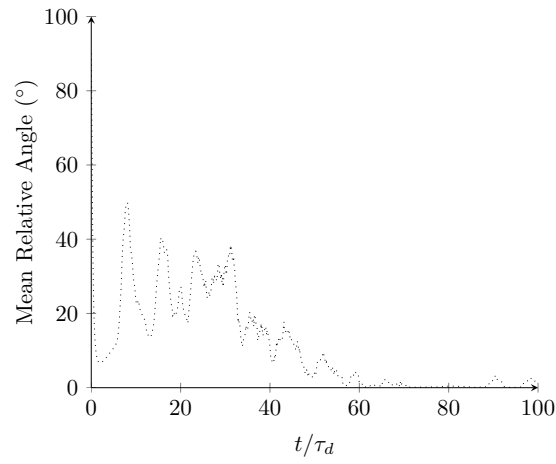


(b) Stokes number of 1.

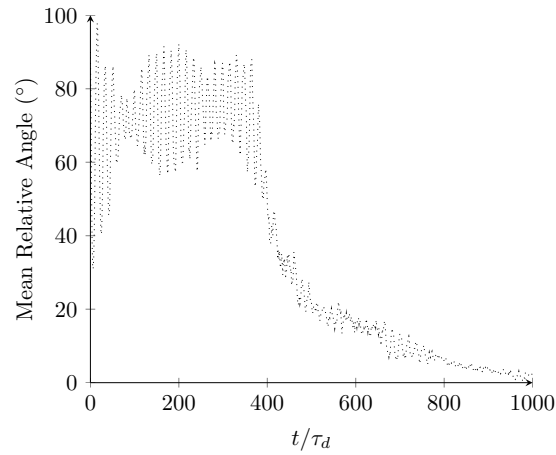


(c) Stokes number of 10.

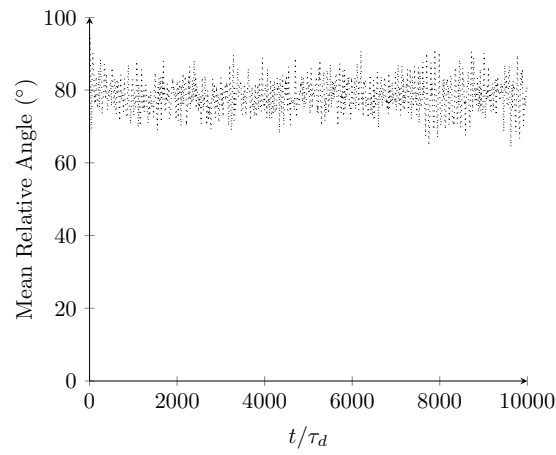
Figure 7.3: Plots of mean relative speed for each velocity component for a range of Stokes numbers.



(a) Stokes number of 0.1.



(b) Stokes number of 1.



(c) Stokes number of 10.

Figure 7.4: Plots of mean relative speed for each velocity component for a range of Stokes numbers.

7.2.1 Run Time Test

A final check on the existing code is to measure the run time. As the code uses first order numerical methods and runs on a GPU the run time of the code as a function of the number of particles should be linear.

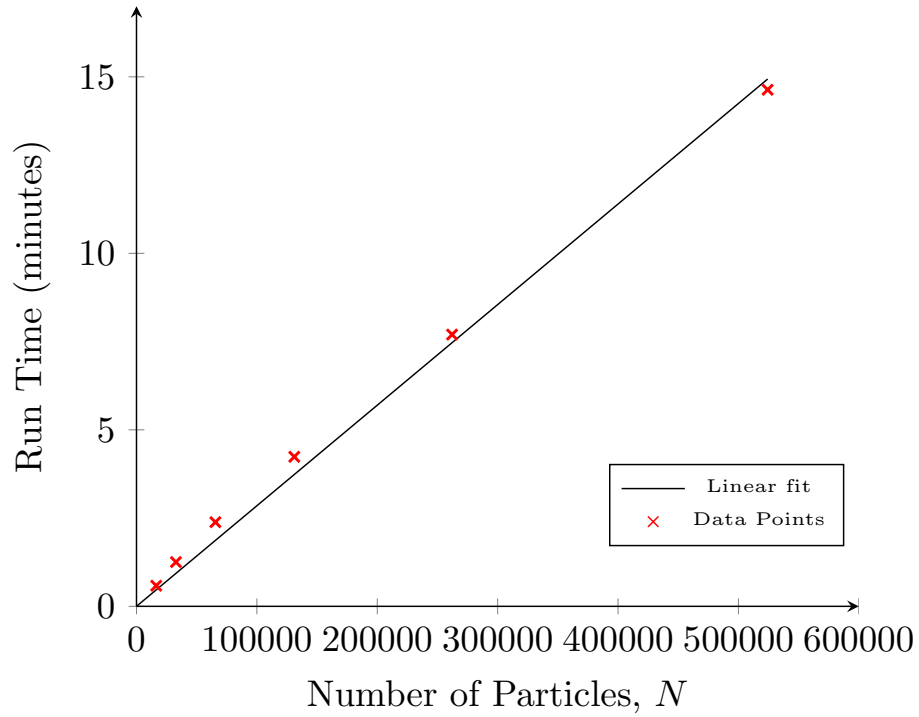


Figure 7.5: Run time of OpenCl code without data logging for different numbers of particles.

Figure 7.5 shows the results from this testing. For low numbers of particles the run time is not linear as the initialisation process is a large percentage of the overall simulation run time. However, for a larger number of particles the run time this is no longer the case.

This set of results provides a point of comparison for when the heat and mass code is added. It should be expected that as first order methods for the heat and mass transfer equations have been used the run time of the code should still be linear.

7.3 Problems Encountered with the Existing Code

Listed in this section are bugs and problems encountered with the existing code. Bugs encountered were found by the author and fixed by E. Andrews. The full history of the code can be found at [20].

7.3.1 Periodic Boundary Condition

One misstep that was made in cutting down the code was removing the boundary condition for the particle location. The simulation domain is cube; when a particle reaches the edge of the boundary its position is reset at the other side of the boundary as shown in Figure 7.6.

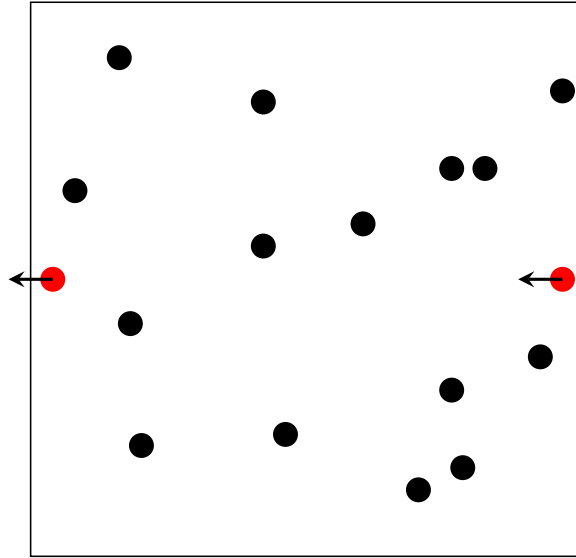


Figure 7.6: Diagram of periodic boundary implementation.

This is a periodic boundary condition and ensures all the particles remain with the cube. Removing this boundary condition means the particles exit the domain at some velocity vector which remains unchanged as it is no longer influenced by the fluid.

7.3.2 Memory Alignment of Structures

The second issue encountered was adding a variable to the particle structure to record the fluid velocity. To ensure consistency between the host code and device code, structures passed to the device should be aligned. This means setting the order of the variables in the structure from the largest data type to the smallest. It also means for using a gcc compiler telling the compiler the required size of the structure. This is the size of the structure that fits within the smallest power of 2. For example, the size of the particle structure with the added variable can be determined from summing the size of the individual data types.

Data Type	Size (bytes)	No. of	Total Size (bytes)
cl_float3	12	5	60
cl_ulong	8	2	16
cl_float	4	4	16

Table 7.4: Data types and their respective sizes for the OpenCL language.

This gives a total size for the structure as 92 bytes. The next highest power of 2 is 128 so this is the size of the structure specified. For MSCV compilers the same value is used but the amount of padding must also be specified. This is the size specified minus the actual size. (In this case 36 bytes).

7.3.3 File Directory Checking

The code makes a number of tests during the setup of the simulation to prevent subsequent errors occurring.

To output the results of the simulation the code writes a text file at some user defined “log step”. For the logging to work the program needs to know an accessible logging directory

exists. It must therefore check this is the case.

When compiled with MSVC the code performs the check using `GetFileAttributes`:

```
if (GetFileAttributes(dir) == INVALID_FILE_ATTRIBUTES) {
    return FALSE;
} else {
    return TRUE;
}
```

When compiled with gcc the code checks if the directory exists by trying to open and close the directory:

```
DIR *dir_obj = opendir(dir);
if (dir_obj) {
    /* Directory exists. */
    closedir(dir_obj);
    return TRUE;
} else if (ENOENT == errno) {
    /* Directory does not exist. */
    return FALSE;
} else {
    /* opendir() failed for some other reason. */
    return FALSE;
}
```

Originally the code used the `stat` structure to perform this check which only worked correctly on some Windows computers.

7.3.4 Initialisation of Particle Speeds

The code uses a Box-Muller transformation to turn uniformly distributed random numbers into a normal distribution for initialisation of particle speeds.

The process is as follows: Generate two random numbers u and v between 0 and 1 using `rand` and `RAND_MAX`:

```
u = (float) rand() / (float) (RAND_MAX);
v = (float) rand() / (float) (RAND_MAX);
```

Next, the normal distribution parameters (mean μ and standard deviation σ) must be specified. Then the normal distribution of speeds using the Box-Muller transform are created using:

$$speed = \sigma \sqrt{-2 \ln(u)} \cos(2\pi v) + \mu \quad (7.7)$$

This formula is problematic in that it contains a log. If u is set to zero then $\ln(u)$ will tend to $-\infty$. This only occurred for two particles in a population of 100,000 particles. This does not affect the other particles but prevented Paraview from being able to read the text files. The fix was to add 1 to the random numbers so that u and v can not be zero:

```
u = (float) (rand() + 1) / (float) (RAND_MAX + 1);
v = (float) (rand() + 1) / (float) (RAND_MAX + 1);
```

7.3.5 Logsteps Being Skipped

The Python implementation of the code logs data at the end of every timestep. This solution is fine for one particle but not suitable when considering thousands of particles as the amount of data generated would be very large. The OpenCL code includes a logstep which determines how frequently data is logged. This means small timesteps can be used to increase accuracy and logsteps can be set larger than the timestep to reduce the amount of data generated.

The code responsible for this is contained within the main time loop of the simulation:

```
if (LOG_DATA && time - last_write >= log_step) {
ret = particlesToHost(queue, gparticles, &hparticles, NUMPART);
printf("Logging at time: %f\n", time);

if (!writeParticles(hparticles, time, prefix, log_dir, NUMPART,
log_vel)) {
return 1;
}

last_write = time;
}
```

The boolean LOG_DATA is used to determine if data is to be logged or not. The second part of the condition is the current time subtract the time data was last written must be greater than or equal to the log step. For some values the second part of the if statement would not return true even when this should have been the case. (This is most likely a rounding error). Putting a "fudge factor" of 0.99 on the logstep as follows:

```
if (LOG_DATA && time - last_write >= 0.99*log_step) {
```

fixed this problem.

8 OpenCL Implementation

The initial testing of numerical methods and the model were performed in Python as this language allows for prototype code to be quickly produced due to its simplicity and feature rich libraries such as NumPy. Python also features capability for creating GPU code through the pyopencl library [21]. However, the existing code uses C and OpenCL to build executable files for running simulations.

8.1 Additions made to the Code

8.1.1 Source Code Structure

For the sake of clarity the existing source code structure is shown in Figure D.1 and can be found in Section D. The new source code structure with DEM functionality removed is shown in Figure 8.1.

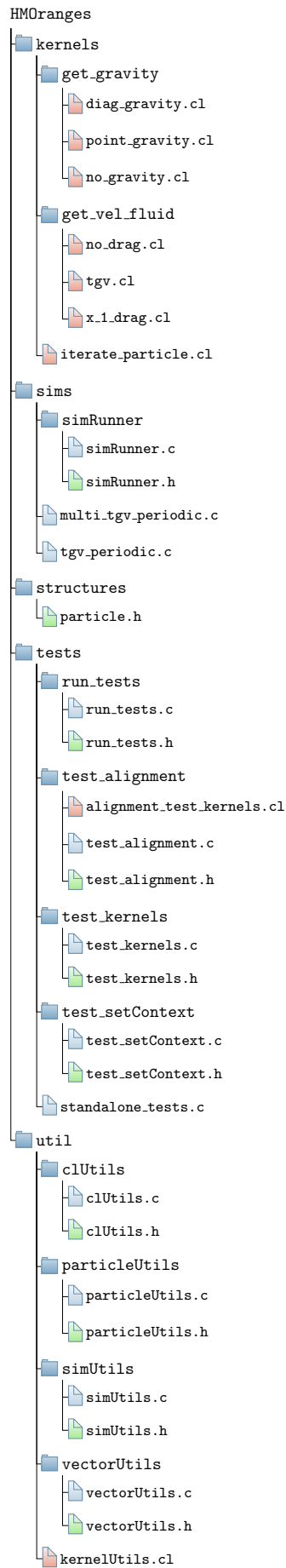


Figure 8.1: File structure of the HMOranges codebase.

8.1.2 Variables

As already mentioned the DEM code had to be stripped out. Once this was completed and results from the code verified the heat and mass transfer methods could be added. This involved adding the following set of new variables to the particle structure:

- T_d;
- W_V;
- T_B;
- L_V;
- C_L;
- P_atm;
- R_bar;
- R;
- W_G;
- theta_1;
- theta_2;
- Y_G;
- Pr_G;
- Sc_G;
- f_2;
- P_G;
- T_G;
- H_deltaT;
- m_d;

In addition to this code was added to the function that initialises the particles so the extra required variables are set to the required initial conditions. Finally, the equations to solve the heat and mass transfer were added to the iterate particle kernel. This included adding a stop condition for updating the particle properties. Namely: position, speed, temperature and mass. When the current particle mass reaches 1% of its starting mass, all properties for the particle are set to zero. At each further timestep the value of the particle properties is logged as zero. This makes later data analysis easier as particles are identified by a row of data in the text file. So the number of rows has to remain the same. This is especially important for utilising Paraview which identifies particles by row.

8.1.3 Heat and Mass Transfer Model

8.2 Verification

Similar to the Python code the OpenCl code must undergo verification tests to ensure the equations are being solved correctly.

8.2.1 Uncoupled Heat Transfer

8.2.2 Uncoupled Mass Transfer

8.2.3 Coupled Heat and Mass Transfer

9 Results

10 Conclusion

10.1 Summary of Results

It has been shown x.

10.2 Future Work

10.2.1 Additional Models and Droplet Species

Firstly, only one model has been implemented. The mass analogy models would make a good starting point for adding further models as this would require minimal alteration to the existing code. This could be taken further in making the code modular enough to support a user being able to easily choose a given model for running a simulation. Further to this a wider range of droplet species could be tested, for example common rocket engine propellants.

10.2.2 Re-ordering of Code to Simplify Running Simulations

This leads into another point of future work. Currently the code must be re-built every time the simulation parameters are changed. A more elegant approach would be to store simulation parameters in a text file which is loaded at runtime. This makes keeping track of simulation parameters easier (they are currently spread out across at least three source code files) and means the executable does not have to be re-built every time the parameters are changed.

10.2.3 Quantitative Analysis and Optimisation of the Code

The code as a whole provides an opportunity for a study on optimisation. One limiting factor is data logging is exceptionally time consuming when compared to running a simulation without data logging. Steps such as finding a faster alternative to C's `fprintf` and perhaps writing in binary instead of to a text file may provide the required performance improvement. Although the choice of numerical scheme has been justified on the basis of simplicity and performance it would be interesting to evaluate higher order implicit methods quantitatively. As fourth order schemes are quite common in the literature.

Appendix

A Method Statement

As this project involves no practical experimentation and only concerns desk-based research there is not a need to submit a risk assessment.

B Derivations

B.1 Velocity of a Particle under Weight and Stokes Drag Forces

The derived ODE from Section 7.1 is:

$$\frac{du}{dt} = g + \frac{1}{\tau_d}(u_f - u) \quad (\text{B.1a})$$

$$\frac{du}{dt} + \frac{u}{\tau_d} = g + \frac{u_f}{\tau_d} \quad (\text{B.1b})$$

An equation of the form $du/dt + Pu = Q$, where P and Q are functions of t . Which can be solved with an integrating factor.

Integrating factor:

$$IF = e^{\int P dt} \quad (\text{B.2a})$$

$$= e^{\int 1/\tau_d dt} \quad (\text{B.2b})$$

$$= e^{t/\tau_d} \quad (\text{B.2c})$$

Therefore:

$$e^{t/\tau_d} \frac{du}{dt} + e^{t/\tau_d} \frac{u}{\tau_d} = e^{t/\tau_d} \left(g + \frac{u_f}{\tau_d} \right) \quad (\text{B.3a})$$

$$\frac{d}{dt} \left(e^{t/\tau_d} u \right) = e^{-t/\tau_d} \left(g + \frac{u_f}{\tau_d} \right) \quad (\text{B.3b})$$

$$e^{t/\tau_d} u = \int e^{t/\tau_d} \left(g - \frac{u_f}{\tau_d} \right) dt \quad (\text{B.3c})$$

$$e^{t/\tau_d} u = \tau_d e^{t/\tau_d} \left(g + \frac{u_f}{\tau_d} \right) + C \quad (\text{B.3d})$$

Solve for C with the boundary condition $u = 0$ when $t = 0$:

$$0 = \tau_d \left(g + \frac{u_f}{\tau_d} \right) + C \quad (\text{B.4a})$$

$$C = -\tau_d \left(g + \frac{u_f}{\tau_d} \right) \quad (\text{B.4b})$$

So that:

$$e^{t/\tau_d} u = \tau_d e^{t/\tau_d} \left(g + \frac{u_f}{\tau_d} \right) - \tau_d \left(g + \frac{u_f}{\tau_d} \right) \quad (\text{B.5a})$$

$$u = \tau_d \left(g + \frac{u_f}{\tau_d} \right) - \tau_d \left(g + \frac{u_f}{\tau_d} \right) e^{-t/\tau_d} \quad (\text{B.5b})$$

$$u = \tau_d \left(g + \frac{u_f}{\tau_d} \right) (1 - e^{-t/\tau_d}) \quad (\text{B.5c})$$

B.2 Backward Euler Method for Temperature ODE

Rearrangement of the backward Euler method for the temperature ODE:

$$T_{d_{n+1}} = T_{d_n} + \Delta t \left[\frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) (T_G - T_{d_{n+1}}) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right] \quad (\text{B.6a})$$

$$\frac{T_{d_{n+1}}}{\Delta t} = \frac{T_{d_n}}{\Delta t} + \left[T_G \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) - T_{d_{n+1}} \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right] \quad (\text{B.6b})$$

$$\frac{T_{d_{n+1}}}{\Delta t} + T_{d_{n+1}} \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) = \frac{T_{d_n}}{\Delta t} + \left[T_G \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right] \quad (\text{B.6c})$$

$$T_{d_{n+1}} \left[\frac{1}{\Delta t} + \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) \right] = \frac{T_{d_n}}{\Delta t} + \left[T_G \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right] \quad (\text{B.6d})$$

$$T_{d_{n+1}} = \frac{\frac{T_{d_n}}{\Delta t} + \left[T_G \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right]}{\left(\frac{1}{\Delta t} + \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) \right)} \quad (\text{B.6e})$$

$$T_{d_{n+1}} = \frac{T_{d_n} + \Delta t \left[T_G \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) + \left(\frac{L_V}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T} \right]}{\left(1 + \Delta t \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) \right)} \quad (\text{B.6f})$$

B.3 Backward Euler Method for Mass ODE

Rearrangement of the backward Euler method for the mass ODE:

$$m_{d_{n+1}} = m_{d_n} + \Delta t \left[-\frac{Sh}{3 Sc_G} \left(\frac{m_{d_{n+1}}}{\tau_d} \right) H_M \right] \quad (\text{B.7a})$$

$$\frac{m_{d_{n+1}}}{\Delta t} = \frac{m_{d_n}}{\Delta t} + \left[-\frac{Sh}{3 Sc_G} \left(\frac{m_{d_{n+1}}}{\tau_d} \right) H_M \right] \quad (\text{B.7b})$$

$$\frac{m_{d_{n+1}}}{\Delta t} + \frac{Sh}{3 Sc_G} \left(\frac{m_{d_{n+1}}}{\tau_d} \right) H_M = \frac{m_{d_n}}{\Delta t} \quad (\text{B.7c})$$

$$m_{d_{n+1}} \left[\frac{1}{\Delta t} + \frac{Sh}{3 Sc_G} \left(\frac{H_M}{\tau_d} \right) \right] = \frac{m_{d_n}}{\Delta t} \quad (\text{B.7d})$$

$$m_{d_{n+1}} = \frac{\frac{m_{d_n}}{\Delta t}}{\frac{1}{\Delta t} + \frac{Sh}{3 Sc_G} \left(\frac{H_M}{\tau_d} \right)} \quad (\text{B.7e})$$

$$m_{d_{n+1}} = \frac{m_{d_n}}{1 + \Delta t \frac{Sh}{3 Sc_G} \left(\frac{H_M}{\tau_d} \right)} \quad (\text{B.7f})$$

B.4 Uncoupled Heat Transfer

Analytic solution of the uncoupled heat transfer ODE:

$$\frac{dT_d}{dt} = \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) (T_G - T_d) \quad (\text{B.8})$$

To make the solution steps clearer, define the constants:

$$A = \frac{f_2 Nu}{3 Pr_G} \left(\frac{\theta_1}{\tau_d} \right) \quad (\text{B.9a})$$

$$B = T_G \quad (\text{B.9b})$$

The ODE can then be solved for $T_{d_{n+1}}$ for a given increment in time Δt from a state where $T_d = T_{d_n}$:

$$\frac{dT_d}{dt} = A(B - T_d) \quad (\text{B.10a})$$

$$\int_{T_{d_n}}^{T_{d_{n+1}}} \frac{dT_d}{(B - T_d)} = \int_0^{\Delta t} A dt \quad (\text{B.10b})$$

$$[\ln(B - T_d)]_{T_{d_n}}^{T_{d_{n+1}}} = [A t]_0^{\Delta t} \quad (\text{B.10c})$$

$$-\ln(B - T_{d_{n+1}}) + \ln(B - T_{d_n}) = A\Delta t \quad (\text{B.10d})$$

$$\ln\left(\frac{B - T_{d_{n+1}}}{B - T_{d_n}}\right) = -A\Delta t \quad (\text{B.10e})$$

$$\frac{B - T_{d_{n+1}}}{B - T_{d_n}} = e^{-A\Delta t} \quad (\text{B.10f})$$

$$B - T_{d_{n+1}} = (B - T_{d_n})e^{-A\Delta t} \quad (\text{B.10g})$$

$$T_{d_{n+1}} = B - (B - T_{d_n})e^{-A\Delta t} \quad (\text{B.10h})$$

Hence, the final analytic solution is:

$$T_{d_{n+1}} = T_G - (T_G - T_{d_n})e^{-\left(\frac{f_2 Nu}{3Pr_G} \left(\frac{\theta_1}{\tau_d}\right)\right)\Delta t} \quad (\text{B.11})$$

B.5 Uncoupled Mass Transfer

The analytic solution for the uncoupled mass transfer ODE:

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} \frac{m_d}{\tau_d} H_M \quad (\text{B.12a})$$

$$\frac{\rho_d \pi D^2}{2} \frac{dD}{dt} = -\frac{Sh}{3Sc_G} \frac{18\rho_d \pi D^3 \mu_g}{6\rho_d D^2} H_M \quad (\text{B.12b})$$

$$D \frac{dD}{dt} = -\frac{2Sh}{Sc_G \rho_d} \mu_g H_M \quad (\text{B.12c})$$

$$\int_{D_n^2}^{D_{n+1}^2} D dD = -\frac{2Sh}{Sc_G \rho_d} \mu_g H_M \int_0^{\Delta t} dt \quad (\text{B.12d})$$

$$\left[\frac{D^2}{2}\right]_{D_n^2}^{D_{n+1}^2} = -\frac{2Sh}{Sc_G \rho_d} \mu_g H_M [t]_0^{\Delta t} \quad (\text{B.12e})$$

$$\frac{1}{2} [D_{n+1}^2 - D_n^2] = -\frac{2Sh}{Sc_G \rho_d} \mu_g H_M \Delta t \quad (\text{B.12f})$$

$$D_{n+1}^2 = D_n^2 - \frac{4Sh}{Sc_G \rho_d} \mu_g H_M \Delta t \quad (\text{B.12g})$$

Solved for mass:

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} \frac{m_d}{\tau_d} H_M \quad (\text{B.13a})$$

$$\frac{dm_d}{dt} = -\frac{Sh}{3Sc_G} H_M m_d^{1/3} (3\mu_G) \left(\left(\frac{6}{\rho_d} \right)^{1/3} \pi^{2/3} \right) \quad (\text{B.13b})$$

$$\frac{dm_d}{dt} = -\frac{Sh}{Sc_G} \mu_G H_M \left(\frac{6}{\rho_d} \right)^{1/3} \pi^{2/3} m_d^{1/3} \quad (\text{B.13c})$$

$$\int_{m_{dn}}^{m_{dn+1}} \frac{dm_d}{(m_d)^{1/3}} = -\frac{Sh}{Sc_G} \mu_G H_M \left(\frac{6}{\rho_d} \right)^{1/3} \pi^{2/3} \int_0^{\Delta t} dt \quad (\text{B.13d})$$

$$\left[\frac{3}{2} (m_d)^{2/3} \right]_{m_{dn}}^{m_{dn+1}} = -\frac{Sh}{Sc_G} \mu_G H_M \left(\frac{6}{\rho_d} \right)^{1/3} \pi^{2/3} [t]_0^{\Delta t} \quad (\text{B.13e})$$

$$\frac{3}{2} \left[(m_{dn+1})^{2/3} - (m_{dn})^{2/3} \right] = -\frac{Sh}{Sc_G} \mu_G H_M \left(\frac{6}{\rho_d} \right)^{1/3} \pi^{2/3} \Delta t \quad (\text{B.13f})$$

$$(m_{dn+1})^{2/3} = (m_{dn})^{2/3} - \frac{2Sh}{3Sc_G} \mu_G H_M \left(\frac{6}{\rho_d} \right)^{1/3} \pi^{2/3} \Delta t \quad (\text{B.13g})$$

As:

$$\tau_d = \frac{\rho_d D^2}{18\mu_g} \quad (\text{B.14})$$

and:

$$D = \left(\frac{6}{\rho_d \pi} \right)^{1/3} (m_d)^{1/3} \quad (\text{B.15})$$

B.6 Terminal Velocity

The particle reaches a terminal velocity when the weight forces balances the drag force:

$$F_g = F_d \quad (\text{B.16})$$

Using the definitions for F_g and F_d from Section x:

$$mg = \frac{m}{\tau_d} (u - v) \quad (\text{B.17a})$$

$$g = \frac{1}{\tau_d} (u - v) \quad (\text{B.17b})$$

$$v = u - g\tau_d \quad (\text{B.17c})$$

$$v = u - g \frac{\rho_d D^2}{18\mu_G} \quad (\text{B.17d})$$

C Physical Data

Note that T_R is the reference temperature used when evaluating a property. This is particularly important to take into account when using the “1/3 rule”. The physical data presented here is as per [2]. The original source of physical data is also referenced.

C.1 Air

Data originally from [22].

$$W_C = 28.97 \text{ kg } (kg \text{ mole})^{-1} \quad (\text{C.1a})$$

$$\mu_C = 6.19 \times 10^{-6} + 4.604 \times 10^{-8} T_R - 1.051 \times 10^{-11} T_R^2 \text{ kg } m^{-1} s^{-1} \quad (\text{C.1b})$$

$$\lambda_C = 3.227 \times 10^{-3} + 8.3894 \times 10^{-5} T_R - 1.9858 \times 10^{-8} T_R^2 \text{ J } m^{-1} s^{-1} K^{-1} \quad (\text{C.1c})$$

$$Pr_C = 0.815 - 4.958 \times 10^{-4} T_R + 4.514 \times 10^{-7} T_R^2 \quad (T_R \leq 600 \text{ K}) \quad (\text{C.1d})$$

$$Pr_C = 0.647 - 5.5 \times 10^{-5} T_R \quad (T_R > 600 \text{ K}) \quad (\text{C.1e})$$

The gas pressure is calculated from the reference temperature and the gas density using the ideal gas law. This requires knowing the gas density, the reference data for this is tabulated below and referenced from [23].

C.2 Water

Data originally from [22].

$$W_V = 18.015 \text{ kg } (kg \text{ mole})^{-1} \quad (\text{C.2a})$$

$$T_B = 3731.15 \text{ K} \quad (\text{C.2b})$$

$$C_{p,V} = 8137 - 37.34 T_R + 0.07482 T_R^2 - 4.956 \times 10^{-5} T_R^3 \text{ J } kg^{-1} K^{-1} \quad (\text{C.2c})$$

$$\mu_V = 4.07 \times 10^{-8} T_R - 3.077 \times 10^{-6} \text{ kg } m^{-1} s^{-1} \quad (\text{C.2d})$$

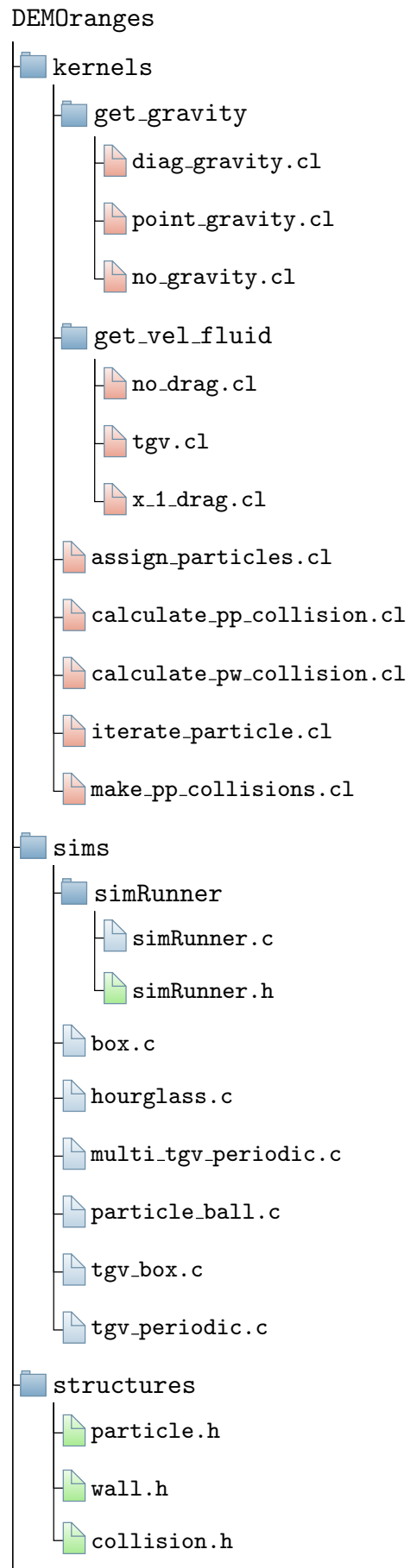
$$L_V = 2.257 \times 10^6 + 2.595 \times 10^3 (371.15 - T_R) \text{ J } Kg^{-1} \quad (\text{C.2e})$$

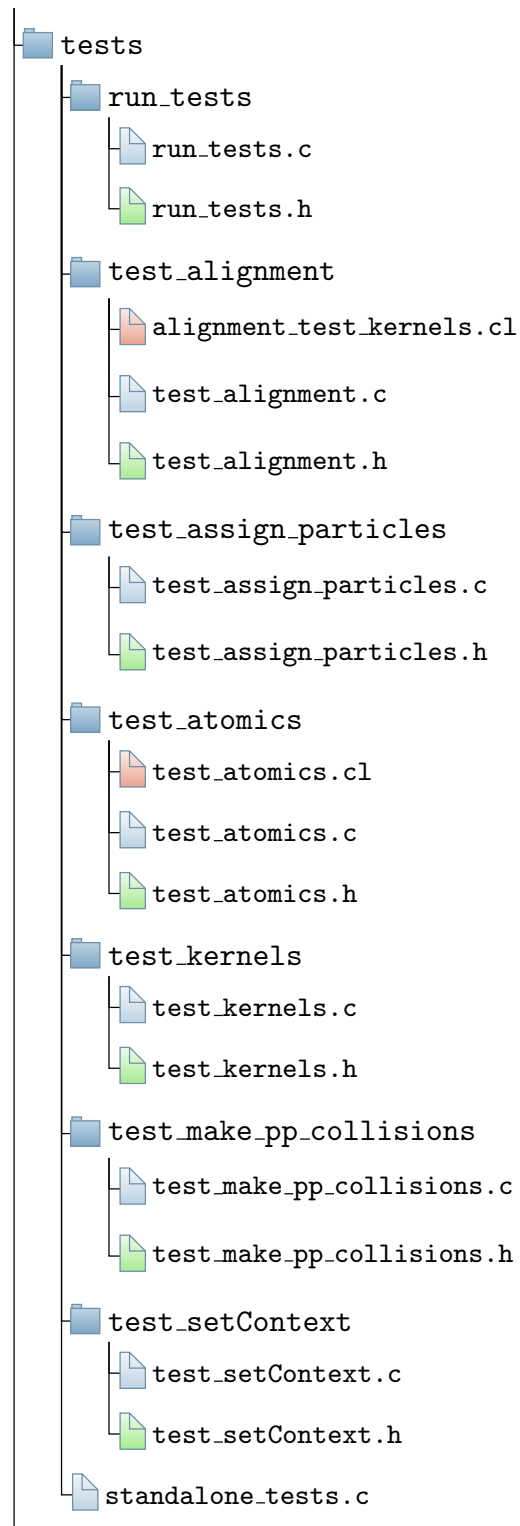
$$\rho_L = 997 \text{ kg } m^{-3} \quad (\text{C.2f})$$

$$C_L = 4148 \text{ J } kg^{-1} K^{-1} \quad (\text{C.2g})$$

$$\lambda_L = 0.6531 \text{ J } m^{-1} s^{-1} K^{-1} \quad (\text{C.2h})$$

D Existing Codebase Structure





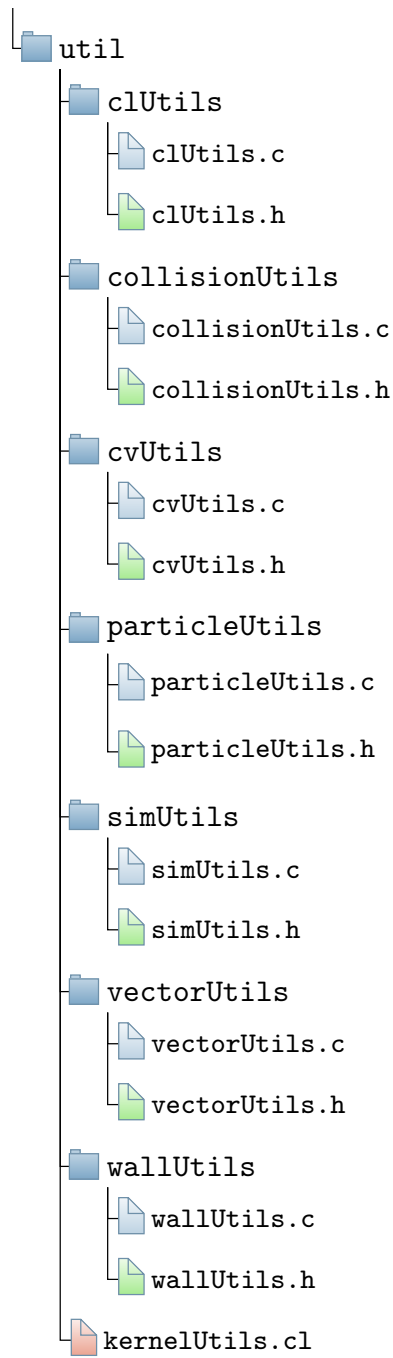


Figure D.1: File structure of the DEMOranges codebase.

References

- [1] E. Andrews, “GPU Enabled Analysis of Agglomeration in Large Particle Populations,” 2018.
- [2] R. S. Miller, K. Harstad, and J. Bellan, “Evaluation of equilibrium and non-equilibrium evaporation models for many-droplet gas-liquid flow simulations,” *International Journal of Multiphase Flow*, vol. 24, pp. 1025–1055, 1998.
- [3] E. K. Shashank, E. Knudsen, and H. Pitsch, “Spray Evaporation Model Sensitivities,” *Annual Research Briefs of the CTR*, pp. 213–224, 2011.
- [4] F. L. Sacomano Filho, G. C. Krieger Filho, J. A. van Oijen, A. Sadiki, and J. Janicka, “A novel strategy to accurately represent the carrier gas properties of droplets evaporating in a combustion environment,” *International Journal of Heat and Mass Transfer*, 2019.
- [5] C. Downing, “The evaporation of drops of pure liquids at elevated temperatures: Rates of evaporation and wet-bulb temperatures,” *American Institute of Chemical Engineers*, vol. 12, pp. 760–766, 1966.
- [6] H. Salman and M. Soteriou, “Lagrangian simulation of evaporating droplet sprays,” *Physics of Fluids - PHYS FLUIDS*, vol. 16, pp. 4601–4622, 12 2004.
- [7] D. Kolaitis and M. Founti, “A Comparative Study of Numerical Models for Eulerian–Lagrangian Simulations of Turbulent Evaporating Sprays,” *International Journal of Heat and Fluid Flow*, vol. 27, pp. 424–435, 06 2006.
- [8] S. Aggarwal, S. William, and A. Tong, “A Comparison of Vaporization Models in Spray Calculations,” *AIAA Journal*, vol. 22, pp. 1448–1457, 09 1984.
- [9] J. Sweet, D. Richter, and D. Thain, “GPU acceleration of Eulerian-Lagrangian particle-laden turbulent flow simulations,” *International Journal of Multiphase Flow*, vol. 99, pp. 437–445, 11 2017.
- [10] Q. Zhang, C. Zhu, and M. Zhu, “Three-Dimensional Numerical Simulation of Droplet Evaporation Using the Lattice Boltzmann Method Based on GPU-CUDA Accelerated Algorithm,” *Communications in Computational Physics*, vol. 23, pp. 1150–1166, 01 2018.
- [11] C. Obrecht, B. Tourancheau, and F. Kuznik, “Performance Evaluation of an OpenCL Implementation of the Lattice Boltzmann Method on the Intel Xeon Phi,” *Parallel Processing Letters*, vol. 25, 09 2015.
- [12] W. Verdier, P. Kestener, and A. Cartalade, “Performance portability of lattice boltzmann methods for two-phase flows with phase change,” unpublished results.
- [13] W. Nazaroff and L. Alvarez-Cohen, *Environmental Engineering Science*. John Wiley and Sons, Inc., 2001.
- [14] R. Deiterding, “Lecture notes on conduction heat transfer,” November 2019.
- [15] J. Shrimpton, *An Introduction to Engineering Thermofluids*. Cuesta Ltd, 2015.
- [16] R. Deiterding, “Lecture notes on convective heat transfer,” November 2019.
- [17] P. W. Atkins and J. D. Paula, *Physical chemistry*. Freeman, 2010.
- [18] A. P. Pinheiro and J. M. Vedovoto, “Evaluation of Droplet Evaporation Models and the Incorporation of Natural Convection Effects,” *Flow, Turbulence and Combustion*, vol. 102, no. 3, p. 537–558, Jan 2018.

- [19] A. Frohn, *Dynamics of Droplets*. Springer, 2000.
- [20] E. Andrews. (2020) DEMOranges Commits. [Online]. Available: <https://github.com/Xorgon/DEMOranges/commits/master>
- [21] pypi. (2019) pyopencl 2019.1.2. [Online]. Available: <https://pypi.org/project/pyopencl/>
- [22] G. M. Harpole, “Droplet Evaporation in High Temperature Environments,” *Journal of Heat Transfer*, vol. 103, no. 1, pp. 86–91, 02 1981. [Online]. Available: <https://doi.org/10.1115/1.3244437>
- [23] Y. A. Çengel, *Fundamentals of thermal-fluid sciences*, 3rd ed. McGraw-Hill, 2008.