Numerical Analysis of Droplet Distribution in Spray Painting

Shyam Sundar Hemamalini
Faculty of Mechanical, Materials and Maritime Engineering, TU Delft

Ravi Ramesh
Faculty of Aerospace Engineering, TU Delft

A Navier-Stokes solver was used to solve a multiphase fluid flow in a narrow, cylindrical pipe. The approach used was a one-way, Euler-Lagrangian one in order to solve the governing equations of the multiphase system. The domain for which the problem was solved was given a uniform mesh, and while the flow was essentially 3D, the circumferential and axisymmetric symmetry of the domain ensured that the flow could be modelled in 2D. As the flow involved two distinct phases, the problem statement was essentially divided into two major parts, one each for the continuous phase and the dispersed phase. First, the solution for the continuous phase alone, which was air in this case, was established, using a 2D Prandtl mixing length based RANS solver, with Nikuradse's formula used to implement the mixing length formulation. Various boundary conditions for calculating velocity and pressure were set up, and the steady state solution obtained to be exported to the dispersed phase. The data was converted from the axisymmetric form to the Cartesian form and interpolated for various control volumes to give the final flow solution for the entire domain. Next, the dispersed phase, which in this case was paint, was modelled by using the discrete eddy model, in tandem with a Monte Carlo Simulation solver. The motion of the particles through the channel was visualised. The plot of the particle diameter at the centerline at the final time step, the distribution of particle diameter at the exit time and exit velocity were obtained from the stored particle values. The distribution of particle position at the exit, exit times for the particles and exit velocity of the particles were presented and discussed. The effect of particle diameters on particle relaxation time and subsequently, the interaction of the particles with eddies were observed and discussed. The scope for improvement for the solver was also discussed in detail.

Nomenclature

D_p	=	Particle diameter [m]
k	=	Turbulent kinetic energy [J/kg]
L_m	=	Mixing length [m]
n	=	Time instant [s]
p	=	Pressure stored at the cell centres [Pa]
r	=	Radial location in the pipe [m]
r_0	=	Radial location corresponding to centerline [m]
Re_p	=	Particle Reynolds Number [-]
T_e	=	Eddy lifetime [s]
T_t	=	Eddy turn-over time [s]
u	=	x-velocity component of the fluid [m/s]
v	=	y-velocity component of the fluid [m/s]
$\overline{u}$	=	Mean component of u-velocity [m/s]
u'	=	Fluctuating component of u-velocity [m/s]
u^*	=	Predicted velocity at a given time step [m/s]
$\vec{U}$, **u**	=	Velocity vector of the continuous phase [m/s]
$\vec{V}$, **v**	=	Velocity vector of the dispersed phase [m/s]
∇	=	Divergence operator
β	=	Relaxation factor [-]
δ	=	Time-stepping multipliers [-]
Δt	=	Time step [s]
ρ	=	Density of the continuous phase [kg/m^3]
ρ_p	=	Density of the dispersed phase [kg/m^3]
μ	=	Dynamic molecular viscosity [kg/m-s]
ν	=	Kinematic molecular viscosity [kg/m-s]
ν_T	=	Kinematic turbulent eddy viscosity [kg/m-s]
τ_p	=	Particle relaxation time [s]

I. Introduction

MULTIPHASE flows are commonly encountered in a wide variety of situations in the industry. This includes oil and gas flows [1], sedimentation [2], bubbly flows [3], biological flows [4] such as the flow of blood and so on. A complete analysis of such flows is crucial in obtaining a better understanding of the physics of the problem statement at hand. The accurate computational simulation of such flows have been a challenge so far. This has been due to various factors that have to be taken into consideration in order to simulate such flows - the disparity in the time scales of the various phases, modelling phenomena at the small scales, such as the interaction between drops, bubbles [5], instabilities such as Rayleigh-Taylor in the case of stratified flows [6], and so on, that affect large scale phenomena such as large scale chemical reactors, energy production systems, systems involving transport of fuels, oil extraction and so on.

Such flows are usually classified based on the type of phases present amongst the continuous phase and the dispersed phase, such as gas-liquid, gas-solid, liquid-liquid, liquid-solid flows, and the concentration/volume fraction of the dispersed phase. Dispersed multiphase flows fall into the category of gas-solid or liquid-solid multiphase flows where the dispersed phase has a very low concentration in the mixture. Dispersed multiphase flows are common in the industry and can be seen in sprays, aerosols, bubbly flows, sedimentation to name some examples.

Numerical simulations of dispersed multiphase flows are known for their complexity and computational difficulty because they are, in essence, transient and three-dimensional. Solutions of such domains are computationally expensive and this has been one of the reasons for meagre development of CFD for such flows compared to those for single phase flows. However, developments in computational architecture and the improvements in performance have made simulations of such flows faster. As a result, there have been major developments in the past decade.

II. Problem Formulation

The problem formulation involves the simulation of a 2D multiphase flow through a narrow, cylindrical pipe. Being a problem that is commonly encountered in many relevant industrial applications, such as microfluidics [7], transport of oil and gas flows, this problem statement was hence, derived from such a problem that could be modified into code and explained in a lucid manner. This involves the division of the flow physics into the continuous phase as well as the dispersed phase for the purpose of simplicity, with the solution of the continuous phase being a necessity for solving for the forces acting on the particles in the dispersed phase owing to the one-way coupling from the continuous phase.

III. Methodologies

Since the domain of interest is a dispersed multiphase flow that is similar to spray painting, certain assumptions can be made, such as:

1) The concentration of paint particles in the mixture is very low.
2) There is negligible interaction between two paint particles. Thus, every particle is independent and does not "see" any other particles in the domain.
3) The collision between the particles and the wall are considered to be elastic. However, the walls of the pipe do not deform upon collision and the pipe stays perfectly cylindrical and rigid.
4) The paint particles are not subjected to internal stresses or deformations. The force acting on the paint particles is entirely converted into the acceleration of the particles.
5) The effect of the solid particles on the fluid is negligible.
6) For the sake of simplicity, the paint particles are of uniform size and mass. They also do not disintegrate or agglomerate during its motion through the pipe.

The above assumptions result in the choice of the following methodologies to be used for the numerical simulation for this problem.

1) Since the interaction between the continuous and the dispersed phase is assumed to be negligible, a one-way coupling from the continuous phase to the dispersed phase would be sufficient.
2) Since the dispersed phase is in the form of rigid particles of fixed shape and mass and are considered as solids, with the continuous phase being a fluid, an Euler-Lagrange type of solver can be used to simulate this case.
3) Since the dispersed phase particles do not interact with each other and are independent of other particles, a simultaneous Monte-Carlo simulation of several particles can be employed.

A. Euler-Lagrange Method

The Euler-Lagrangian method involves the fluid phase being solved in the conventional Eulerian averaged method, while the particle motion is solved in a Lagrangian fashion [8]. In order to implement this approach for multiphase flow, the velocity is computed at each time to carry out the Lagrangian step for each phase, in which case the velocity plays a dual role of being a momentum density, as well as an agent for convection in the recalculation step.

The governing equations involved in this method include both the continuity as well as the momentum equations. The continuous form of the continuity equation is given by Equation 1.

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

The continuous form of the Navier-Stokes equation is given by Equation 2.

$$\rho \frac{\partial \mathbf{u}}{\partial t} + \rho(\nabla \cdot \mathbf{u})\mathbf{u} = \nabla p + \rho \mathbf{g} + \mu_0 \nabla^2 \mathbf{u} \tag{2}$$

B. One-way Phase Coupling

One-way phase coupling involves solving a single phase and then giving the velocities of that phase as an input to the equation of motion of the particles. This method assumes that the concentration of the dispersed phase is much lesser than that of the continuous phase in the domain. Thus, the concentration of particles of the dispersed phase in a multiphase flow is, say, less than 0.01% by volume of that of the continuous phase [9], which causes the particles to have a very little effect on the motion of the continuous phase. This means that the continuous phase is responsible for changing the particle energy and velocity, and makes the analysis of such multiphase flows much easier.

In general, coupling can take place through the transfer of mass, momentum and energy between phases [10]. These include:

1) *Mass coupling* involves the addition or the removal of mass by evaporation or condensation respectively. This can be verified by calculating the mass coupling parameter, which is defined by Equation 3.

$$\Pi_{mass} = \frac{\dot{M}_d}{\dot{M}_c} \tag{3}$$

where $\dot{M}_d$ and $\dot{M}_c$ is the mass generated by the dispersed phase and the continuous phase respectively. For the given problem, the dispersed phase generates much lesser mass in comparison to continuous phase, Π_{mass} is negligible, and hence, mass coupling can be neglected for the given problem statement.

2) *Momentum coupling* is caused by the result of drag force on the continuous and the dispersed phases. This can also be due to the momentum addition or depletion due to mass transfer. This is definitely an important factor to be taken into consideration in the current problem statement, which can be defined by the momentum coupling parameter, given by Equation 4.

$$\Pi_{mom} = \frac{F_D}{Mom_c} \tag{4}$$

where F_D is the Stokes drag associated with the dispersed phase, which is given by $\mathbf{F_D} = nL^3 3\pi\mu_c D(\mathbf{u} - \mathbf{v})$. Because of the fact that the current problem statement involves particles that generate a lot of Stokes drag, $\Pi_{mom} \gg 1$. Hence, momentum coupling cannot be neglected for the given problem statement.

3) *Energy coupling* occurs when there is a heat transfer between phases. This includes the transfer of both thermal as well as kinetic energy. Like the mass and momentum coupling parameters, an equivalent energy coupling parameter can be introduced, which can be defined by Equation 5.

$$\Pi_{ener} = \frac{\dot{Q}_d}{\dot{E}_c} \tag{5}$$

where $\dot{Q}_d$ is the heat transferred associated with the dispersed phase elements, defined by $\dot{Q}_d = nL^3 \pi Nu k_c' D(T_d - T_c)$, where Nu is Nusselt's number and k_c' is the thermal conductivity of the continuous phase, and E_c is the energy flux of the continuous phase. In the given problem statement, because there is no heat transferred between the dispersed phase and continuous phase, Π_{ener} can be neglected and hence, energy coupling can be ignored.

C. Monte-Carlo Statistical Method

The Monte-Carlo method has been used in a variety of contexts in order to obtain the behaviour of a sample distribution of data when sampling is performed randomly [11]. From an engineering point of view, this method has been used quite frequently in the past in order to obtain the solution for large complex systems, where approximations that can be done analytically [12]. Broadly speaking, there are mainly two different kinds of such methods that can be used to analyze the flow field in a turbulent system, namely the Eulerian approach as well as the Lagrangian approach. While the former utilizes a velocity field that is generated over a given spatial domain, the latter bases itself on the construction of the evolution of the particles in the system and not on the velocity field itself. Thus, the evolution of turbulence is performed by simulating the motion of the particles themselves rather than using an initial reference velocity field [13]. In the context of multiphase flows in particular, Monte-Carlo algorithms are used to generate the components of velocity fluctuations that are used to calculate the turbulent eddy viscosity and velocity gradients. Furthermore, in this case, the kind of Monte Carlo simulations is decided by the mean free path (λ). For dilute flows, the direct simulation Monte Carlo (DSMC) can be used, where the ratio of the mean free path to the diameter (λ/d) $\geq$ 10. In such a case, the major steps involved in a general Monte-Carlo algorithm are as follows:

1) Using a probability distribution function (PDF), random values are generated for an input variable.
2) An analysis is performed to check if the solution lies within the mathematical realm of probability.
3) Repeating steps 1. and 2., counting the number of random values that lie within the range of acceptable values for fluctuations in the quantity for the given application.
4) The mean probability (or expectation) of the occurrence of the event is then obtained by the relation given by:

$$p_e = \frac{N_e}{N}$$

Where N_e is the probability of the occurrence of the event, and N is the total number of samples.

The random fluctuations obtained from the PDFs depend on the kind of the PDF used, including a normal distribution. The significance of this algorithm for the given problem statement lies in the fact that it is easy to implement for further analysis, including computing the velocity fluctuations in all directions by using the relations provided by the Discrete Eddy Model, as explained in Section V.C.

One of the advantages of this method is the confidence that one can obtain as the number of data samples is increased, especially when the computational cost is not very significant. Another advantage of this approach is the fact that the generation of disordered velocity fields in turbulent flows, and simulating particle trajectories in particle-laden flows, the latter being especially significant for multiphase flows.

D. Computational Tools Used

All the simulations and the post-processing analyses were executed using Python 3.7.10 in Google Colaboratory (non-Pro version). The Python runtime server is hosted by Google and runs on a virtual machine running Ubuntu 18.04, powered by an Intel Xeon (2x hyper-threaded) 2.30 GHz processor paired with 13GB RAM and 70GB of available hard-disk space, with a 12 hour timeout window for continuous executions and a 90 minute idling time.

IV. Simulation of the Continuous Phase

Since a one-way coupling between the continuous and the dispersed phase is to be implemented, the simulation of the two phases can be carried out independently. Since the simulation of the dispersed phase requires flow data of the continuous phase, the simulation of the continuous phase is first executed.

A. Geometry & Mesh

The geometry in consideration is a cylindrical pipe of an arbitrary length and diameter of 50mm and 20mm respectively. The geometry is intrinsically 3-dimensional. However, the flow is assumed to be axisymmetric with the effects of gravity being neglected. This essentially means that any arbitrary half-cross-section of the pipe need only be simulated. Hence, the geometry is now reduced to 2-dimensional with a symmetry boundary at either the top or the bottom of the domain depending on choice of orientation.

A uniform mesh was chosen for its simplicity. Since the flow effects are more well-defined in the radial direction than the streamwise direction, the mesh cell sizing was smaller in the radial direction, at 0.5mm whereas the same in the streamwise direction was 2.5mm. A representative figure showing the geometry and the mesh is presented in Figure 1.

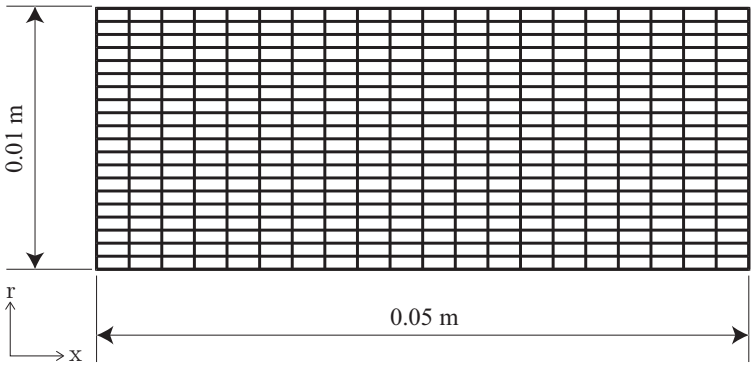

Fig. 1: Geometry and mesh used for the simulation of continuous phase

A staggered grid for velocity and pressure is chosen. This implies that the pressure values are stored at the cell centers of each cell whereas the velocity values are stored at the face centers of each cell, with the components normal to the face in which they are stored in. The fluid properties such as the density, the kinematic viscosity and the turbulent property of eddy viscosity are stored in the cell centers with the pressure. A pictorial representation of the staggered grid arrangement used in the simulation is as shown in Figure 2.

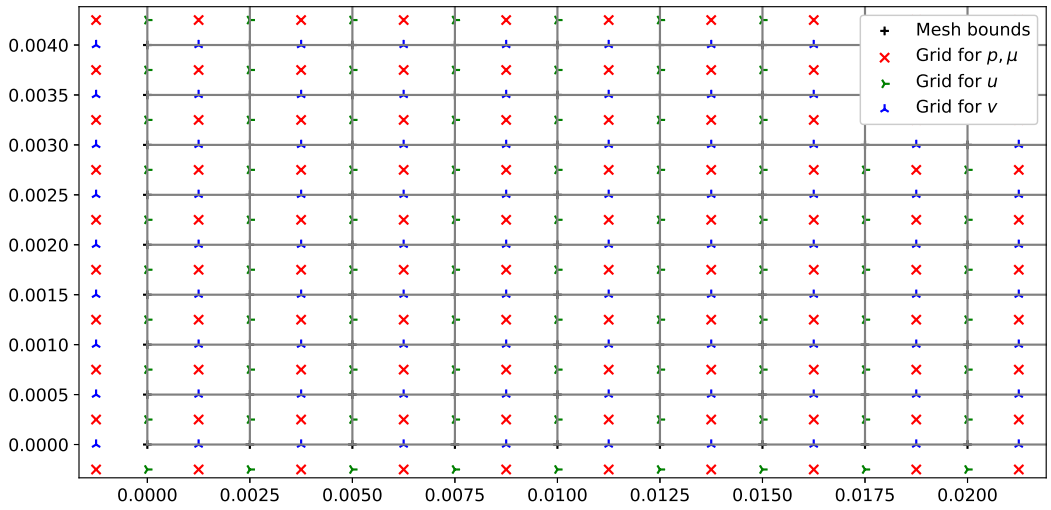

Fig. 2: Staggered grid arrangement used for the simulation of continuous phase

B. Discretisation of Navier-Stokes Equations

The discretization of the Navier-Stokes equations is carried out in the following steps:

1) The velocity field is calculated by ignoring the pressure, and then the pressure is added. This step is similar to the previous velocity-pressure formulation as given in Equation 2, but the advection (**A**, say) and diffusion (**D**, say) terms are separated out as shown in Equation 6 in the predictor step.

$$\frac{\mathbf{u}^* - \mathbf{u}^\mathbf{n}}{\Delta t} = -\mathbf{A}^\mathbf{n} + \mathbf{g}^\mathbf{n} + \frac{1}{\rho^n}\mathbf{D}^\mathbf{n} \tag{6}$$

2) The pressure is introduced in the second step as given in Equation 7 as the corrector step.

$$\frac{\mathbf{u}^{\mathbf{n+1}} - \mathbf{u}^*}{\Delta t} = -\frac{\nabla_h p}{\rho^n} \tag{7}$$

Where ∇_h is the discrete gradient operator. On adding Equations 6 and 7, one obtains the discrete form of the Navier-Stokes equations.

3) In order to obtain the pressure field, one must take the divergence of Equation 7 and account for the divergence free velocity field $\mathbf{u^{n+1}}$, as given in Equation 8.

$$\nabla_h \cdot \left(\frac{1}{\rho^n}\nabla_h p\right) = \frac{1}{\Delta t}\nabla_h \cdot \mathbf{u}^* \tag{8}$$

4) The velocities in Equations 6 and 7 are computed by using the predictor-corrector scheme. That involves the computation of the predicted velocity $\mathbf{u}^*$ at each grid position. The horizontal velocity component for the predictor step is given in Equation 9.

$$u^*_{i+1/2,j} = u^n_{i+1/2,j} + \Delta t\left((-A_x)^n_{i+1/2,j} + (g_x)^n_{i+1/2,j} + \frac{1}{\frac{1}{2}(\rho^n_{i+1,j} + \rho^n_{i,j})}(D_x)^n_{i+1/2,j}\right) \tag{9}$$

5) The vertical velocity component for the predicted velocity is given in Equation 10.

$$v^*_{i,j+1/2} = v^n_{i,j+1/2} + \Delta t\left((-A_y)^n_{i,j+1/2} + (g_y)^n_{i,j+1/2} + \frac{1}{\frac{1}{2}(\rho^n_{i,j+1} + \rho^n_{i,j})}(D_y)^n_{i,j+1/2}\right) \tag{10}$$

It is to be noted that the density ρ is not known at locations (i + 1/2, j) and (i, j + 1/2) is evaluated by using interpolation, as given by $\rho_{i+1/2,j} = \frac{1}{2}(\rho_{i,j} + \rho_{i+1,j})$

6) The advection terms are discretized by computing them at the cell centers using the finite difference method as mentioned earlier. The advection operator can be discretized separately in the x and y directions as given by Equations 11 and 12 respectively.

$$(A_x)_{i+1/2,j} = \frac{1}{\Delta x \Delta y}([(uu)_{i+1,j} - (uu)_{i,j}]\Delta y + [(uv)_{i+1/2,j+1/2} - (uv)_{i+1/2,j-1/2}]\Delta x) \tag{11}$$

$$(A_y)_{i+1/2,j} = \frac{1}{\Delta x \Delta y}([(uv)_{i+1/2,j+1/2} - (uv)_{i-1/2,j+1/2}]\Delta y + [(vv)_{i,j+1} - (vv)_{i,j}]\Delta x) \tag{12}$$

In the case of staggered grids, however, the above formulation will have to be changed, as the velocities would have to be obtained on integer grid points. These have to be obtained from linear interpolation. Hence, the modified advection equations in the x and y directions are given in the form as shown in Equations 13 and 14 respectively.

$$\begin{aligned}(A_x)^n_{i+1/2,j} = {} & \frac{1}{\Delta x}\left[\left(\frac{u^n_{i+3/2,j} + u^n_{i+1/2,j}}{2}\right)^2 - \left(\frac{u^n_{i+3/2,j} + u^n_{i+1/2,j}}{2}\right)^2\right] \\ & + \frac{1}{\Delta y}\left[\left(\frac{u^n_{i+1/2,j+1} + u^n_{i+1/2,j}}{2}\right)\left(\frac{v^n_{i+1,j+1/2} + v^n_{i+1,j+1/2}}{2}\right) - \left(\frac{u^n_{i+1/2,j} + u^n_{i+1/2,j-1}}{2}\right)\left(\frac{v^n_{i+1,j-1/2} + v^n_{i,j-1/2}}{2}\right)\right]\end{aligned} \tag{13}$$

$$\begin{aligned}(A_y)^n_{i,j+1/2} = {} & \frac{1}{\Delta x}\left[\left(\frac{u^n_{i+1/2,j} + u^n_{i+1/2,j+1}}{2}\right)\left(\frac{v^n_{i,j+1/2} + v^n_{i+1,j+1}}{2}\right) - \left(\frac{u^n_{i-1/2,j+1} + u^n_{i-1/2,j}}{2}\right)\left(\frac{v^n_{i,j+1/2} + v^n_{i-1,j+1/2}}{2}\right)\right] \\ & + \frac{1}{\Delta y}\left[\left(\frac{v^n_{i,j+3/2} + v^n_{i,j+1/2}}{2}\right)^2 - \left(\frac{v^n_{i,j+1/2} + v^n_{i,j-1/2}}{2}\right)^2\right]\end{aligned} \tag{14}$$

7) The diffusion terms can be written in a similar manner to their advection counterparts, i.e., term by term, by using a finite difference scheme to specify the velocity gradients along the boundaries of each control volume. This can be understood better by using Equations 15 and 16 respectively.

$$(D_x)^n_{i+1/2,j} = \frac{\mu_{i,j}}{\Delta x \Delta y}\left[\left(\left.\frac{\partial u}{\partial x}\right|_{i+1,j} - \left.\frac{\partial u}{\partial x}\right|_{i,j}\right)\Delta y + \left(\left.\frac{\partial u}{\partial y}\right|_{i+1/2,j+1/2} - \left.\frac{\partial u}{\partial y}\right|_{i-1/2,j+1/2}\right)\Delta x\right] \tag{15}$$

$$(D_y)^n_{i+1/2,j} = \frac{\mu_{i,j}}{\Delta x \Delta y}\left[\left(\left.\frac{\partial v}{\partial x}\right|_{i+1/2,j+1/2} - \left.\frac{\partial v}{\partial x}\right|_{i-1/2,j+1/2}\right)\Delta y + \left(\left.\frac{\partial v}{\partial y}\right|_{i,j+1} - \left.\frac{\partial v}{\partial y}\right|_{i,j}\right)\Delta x\right] \tag{16}$$

The velocity derivatives in the above formulation are found by using the second order derivatives, i.e., by using a central differencing approach as given by Equations 17 and 18 respectively.

$$(D_x)^n_{i+1/2,j} = \mu_0 \left[\left(\frac{u^n_{i+3/2,j} - 2u^n_{i+1/2,j} + u^n_{i-1/2,j}}{\Delta x^2} \right) + \left(\frac{u^n_{i+1/2,j+1} - 2u^n_{i+1/2,j} + u^n_{i+1/2,j-1}}{\Delta y^2} \right) \right] \tag{17}$$

$$(D_y)^n_{i,j+1/2} = \mu_0 \left[\left(\frac{v^n_{i+1,j+1/2} - 2v^n_{i,j+1/2} + v^n_{i-1,j+1/2}}{\Delta x^2} \right) + \left(\frac{v^n_{i+1,j+3/2} - 2v^n_{i,j+1/2} + v^n_{i,j-1/2}}{\Delta y^2} \right) \right] \tag{18}$$

8) The pressure equations are first solved by obtaining the corrected velocity and substituting that in the continuity equation, as shown in Equation 19.

$$\frac{1}{\Delta x} \left[u^*_{i+1/2,j} - \frac{\Delta t}{\frac{1}{2}(\rho^n_{i+1,j} + \rho^n_{i,j})} \left(\frac{p_{i+1,j} - p_{i,j}}{\Delta x} \right) - u^*_{i-1/2,j} + \frac{\Delta t}{\frac{1}{2}(\rho^n_{i,j} + \rho^n_{i-1,j})} \left(\frac{p_{i,j} - p_{i-1,j}}{\Delta x} \right) \right]$$
$$+ \frac{1}{\Delta y} \left[v^*_{i,j+1/2} - \frac{\Delta t}{\frac{1}{2}(\rho^n_{i,j+1} + \rho^n_{i,j})} \left(\frac{p_{i+1,j} - p_{i,j}}{\Delta y} \right) - v^*_{i,j-1/2} + \frac{\Delta t}{\frac{1}{2}(\rho^n_{i,j} + \rho^n_{i,j-1})} \left(\frac{p_{i,j} - p_{i,j-1}}{\Delta y} \right) \right] = 0 \tag{19}$$

This can also be simplified by multiplying Equation 19 by $-\Delta x \Delta y (2\Delta t)^{-1}$, which gives the formulation as shown in Equation 20.

$$\frac{1}{2\Delta t} \left[\frac{u^*_{i+1/2,j} - u^*_{i-1/2,j}}{\Delta x} + \frac{v^*_{i,j+1/2} - v^*_{i,j+1/2}}{\Delta y} \right] = \frac{1}{\Delta x^2} \left(\frac{p_{i+1,j} - p_{i,j}}{\rho^n_{i+1,j} + \rho^n_{i,j}} - \frac{p_{i,j} - p_{i-1,j}}{\rho_{i,j^n} + \rho^n_{i-1,j}} \right)$$
$$+ \frac{1}{\Delta y^2} \left(\frac{p_{i,j+1} - p_{i,j}}{\rho^n_{i,j+1} + \rho^n_{i,j}} - \frac{p_{i,j} - p_{i,j-1}}{\rho^n_{i,j} + \rho^n_{i,j-1}} \right) \tag{20}$$

Next, the successive over-relaxation (SOR) method is implemented, in which an under-relaxation factor β which is greater than unity is introduced in the discretized version of the governing equations for pressure as shown in Equation 21.

$$p^n_{i,j} = \beta \left[\frac{1}{\Delta x^2} \left(\frac{1}{\rho^n_{i+1,j} + \rho^n_{i,j}} + \frac{1}{\rho^n_{i-1,j} + \rho^n_{i,j}} \right) + \frac{1}{\Delta y^2} \left(\frac{1}{\rho^n_{i,j+1} + \rho^n_{i,j}} \right) \right]^{-1}$$
$$\left[\frac{1}{\Delta x^2} \left(\frac{p^n_{i+1,j}}{\rho^n_{i+1,j} + \rho^n_{i,j}} + \frac{p^n_{i-1,j}}{\rho^n_{i-1,j} + \rho^n_{i,j}} \right) + \frac{1}{\Delta y^2} \left(\frac{p^n_{i,j+1}}{\rho^n_{i,j+1} + \rho^n_{i,j}} + \frac{p^n_{i,j-1}}{\rho^n_{i,j-1} + \rho^n_{i,j}} \right) \right.$$
$$\left. - \frac{1}{2\Delta t} \left(\frac{u^*_{i+1/2,j} - u^*_{i-1/2,j}}{\Delta x} + \frac{v^*_{i,j+1/2} - v^*_{i,j-1/2}}{\Delta y} \right) \right] + (1 - \beta) p^n_{i,j} \tag{21}$$

This method can be made more accurate by computing the residuals at each time step, however, it is also more computationally expensive to do so. For the given problem statement, the SOR method works quite well, and hence, the residuals need not be computed.

9) Once the pressure iterations are converged to a suitable order of magnitude, the velocities at time step n+1 can be determined, assuming that forward differences are used, as shown by Equation 22 and 23 respectively.

$$u^{n+1}_{i+1/2,j} = u^*_{i+1/2,j} - \frac{\Delta t}{\frac{1}{2}(\rho^n_{i+1,j} + \rho_{i,j})} \frac{p_{i+1,j} - p_{i,j}}{\Delta x} \tag{22}$$

$$v^{n+1}_{i,j+1/2} = v^*_{i,j+1/2} - \frac{\Delta t}{\frac{1}{2}(\rho^n_{i,j+1} + \rho_{i,j})} \frac{p_{i,j+1} - p_{i,j}}{\Delta y} \tag{23}$$

C. Initial Conditions & Boundary Conditions

The domain is initialised with $\mathbf{u} = 0$ and $p = 10^5$ Pa. The density in the domain is kept constant at $\rho = 1.25$ kg/m^3. The dynamic viscosity is also kept constant at $\mu = 10^{-5}$ Pa-s. The standard set of boundary conditions - velocity inlet and pressure outlet - are used to solve for the continuous phase. A representative figure is shown in Figure 3.

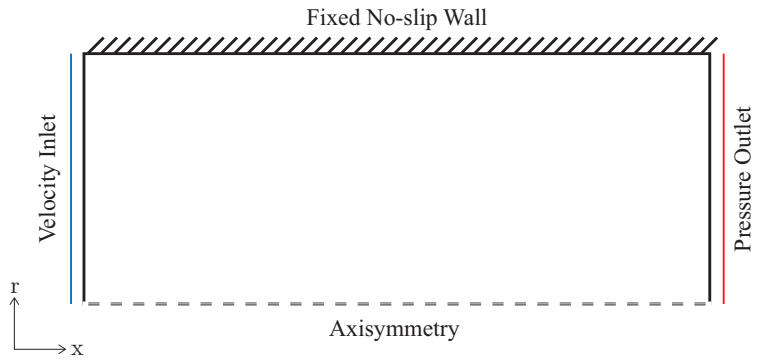

Fig. 3: Boundary conditions used for the simulation of continuous phase

The implications of the chosen boundary condition on the grid of the flow properties such as velocity, pressure and eddy viscosity and the fluid properties such as the kinematic viscosity and the density are discussed below.

Velocity Boundary Conditions

- A *fixed velocity* Dirichlet boundary condition with $u = 5$ m/s and $v = 0$ are specified at the inlet. The imposition that $v = 0$ requires the presence of ghost cells at the inlet for v. The value of u is set directly at the boundary, thus eliminating the need for ghost cells.
- A *zero gradient* Neumann boundary condition with $\frac{\partial \mathbf{u}}{\partial x} = 0$ is imposed at the outlet. This implies that ghost cells are required for both u and v at the outlet to impose the condition that the gradient of velocity in the streamwise direction is zero.
- *A no-slip* boundary is imposed at the wall at the top of the domain. This is similar to the inlet boundary with a Dirichlet boundary condition of $\mathbf{u} = 0$. No ghost cells are required for v since the grid for v is exactly on the boundary. However, ghost cells are required for u.
- A *symmetry* boundary condition is imposed at the bottom wall such that $v = 0$ and $\frac{\partial u}{\partial y} = 0$. Similar to the no-slip boundary, ghost cells are required for u whereas the condition for v can be imposed directly on the grid.

Pressure Boundary Conditions

- A *zero gradient* Neumann boundary condition for pressure is imposed at the inlet, with $\frac{\partial p}{\partial x} = 0$.
- A *fixed value* Dirichlet boundary condition for pressure with a value of $p = 10^5$ Pa is imposed at the outlet.
- Since the bottom boundary is a *symmetry* boundary, a similar condition to that of velocity is imposed at this boundary, i.e., $\frac{\partial p}{\partial y} = 0$.
- At the top boundary, which is a *no-slip* wall, the method described by Tryggvason is followed [8]. In the discretised equations (Equations 19 and 20) for cells close to the top wall, the pressure term should be non-existent. In order to achieve this, a very high density is prescribed at ghost cells above the top boundary. This would diminish the effects of the pressure term in that particular direction, thereby numerically eliminating the term from the equations.

Since pressure is stored at the centers of each cell, ghost cells at each boundary are required for the implementation of the boundary conditions.

D. Solution of Laminar Flow

The implementation of boundary conditions and the discretisation of the Navier-Stokes equations are tested with a simple solution of laminar flow. The pressure corrector algorithm is followed for the simulation, the explanation of which is as follows.

1) The variables p,u and v are initialised based on the conditions stated in IV.C.
2) From Equations 9 and 10, the predictor velocity is determined from the values of velocity at the previous time-step. An example code to determine the horizontal component of the predictor velocity is given in Code Snippet 1.

```
for i in range(1,nx+1):
    for j in range(1,ny+1):
        #Advection terms - x component
        Ax = ((u[i+1,j] + u[i,j])**2 - (u[i,j]+u[i-1,j])**2)/(4*h_x) +\
             ((u[i,j+1] + u[i,j])*(v[i+1,j] + v[i,j]) - (u[i,j] + u[i,j-1])*\
               (v[i+1,j-1] + v[i,j-1]))/(4*h_y)
        #Diffusion terms - x component
        Dx = 0.5*(mu[i+1,j] + mu[i,j])*\
            ((u[i+1,j] -2*u[i,j] + u[i-1,j])/(h_x**2) +\
             (u[i,j+1] -2*u[i,j] + u[i,j-1])/(h_y**2))
        #Predictor step - x component
        ustar[i,j] = u[i,j] + dt*(-Ax + (2/(rho[i,j] + rho[i+1,j]))*Dx + gx)
```

Code Snippet 1: Determination of predictor velocity - horizontal component

3) The continuity equation is then iteratively solved to determine the value of pressure at each cell for the next time step. This is done by solving Equation 19 using the successive over relaxation (SOR) method with a relaxation factor of 1.2. The difference in pressure between each iteration is chosen to be the convergence criteria for the iterations. Once the maximum residual drops below a value of 1e−9 Pa, the solution is assumed to be converged. If the solution does not converge after 1000 iterations, the solution after the 1000th iteration is assumed to be the converged solution. The code highlighting this method is given in Code Snippet 2.

```
iter=0
    while True:
        pn=p.copy()
        iter=iter+1
        for i in range(1,nx+1):
            for j in range(1,ny+1):
                #Pressure-predictor step
                p[i,j]=(1.0-beta)*p[i,j]+beta*p2[i,j]*(\
                    (1./h_x)*( p[i+1,j]/(h_x*(rt[i+1,j]+rt[i,j]))+\
                    p[i-1,j]/(h_x*(rt[i-1,j]+rt[i,j])))+\
                    (1./h_y)*( p[i,j+1]/(h_y*(rt[i,j+1]+rt[i,j]))+\
                    p[i,j-1]/(h_y*(rt[i,j-1]+rt[i,j])))-p1[i,j])

        if np.abs(pn-p).max()<maxerror:
            converged = "Converged"
            break
        if iter==maxiter:
            converged = "Not converged"
            break
```

Code Snippet 2: Iterative solution for pressure using SOR

4) When the pressure at the next time-step is determined, the corrector step can then be applied to determine the velocity field at the next time-step, using Equations 22 and 23. The relevant code for this step is given in Code Snippet 3.

```
#Corrector step
for i in range(1,nx+1):
        for j in range(1,ny+1):
            u[i,j]=ustar[i,j]-dt*(2.0/h_x)*(p[i+1,j]-p[i,j])/(rho[i+1,j]+rho[i,j])

    for i in range(1,nx+1):
        for j in range(1,ny):
            v[i,j]=vstar[i,j]-dt*(2.0/h_y)*(p[i,j+1]-p[i,j])/(rho[i,j+1]+rho[i,j])
```

Code Snippet 3: Velocity correction

5) Now, the pressure and the velocity fields at the boundary pertaining to the Neumann boundary condition are updated as per the condition. The Dirichlet boundaries are kept as is as they are static boundaries. The CFL criterion is applied to determine the time-step for the next time iteration based on the value of the maximum velocity. A CFL number of 0.05 was found to provide stability. This is executed using the code shown in Code Snippet 4.

```
# Inlet dynamic boundary
v[0,:] = -1*v[1,:]
u[0,:] = u[1,:]

# Outlet dynamic boundary
v[-1,:] = 1*v[-2,:]
u[-1,:] = u[-2,:]

# Bottom wall dynamic boundary
u[:,0] = 1*u[:,1]
p[:,0] = 1*p[:,1]

# Top wall dynamic boundary
u[:,-1] = -1*u[:,-2]

# CFL Criterion
dt = CFL/((u.max()/h_x) + (v.max()/h_y))
```

Code Snippet 4: Determination of dt from CFL criterion

6) Steps 2,3,4,5 are repeated considering the newly determined solution at a flow time *n+1* as the values of velocity and pressure at time step *n*. The simulation is carried out until a flow time of 0.5s is reached, at which point the simulation is stopped and the values of velocity and pressure are exported.

The control flow of the program is shown as a flowchart in Figure 4.

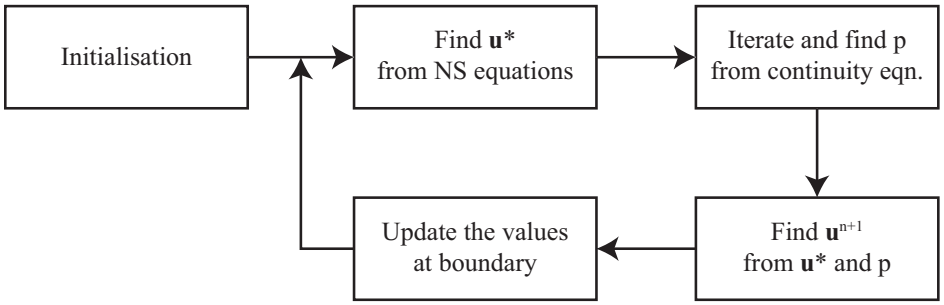

Fig. 4: Control flow for simulation of laminar pipe flow

The obtained values of velocity and pressure at $t = 0.5$s are considered as the initial conditions for the simulation of turbulent flow.

E. Turbulence Model

The turbulence model used in the given problem statement is that of the 2D Prandtl mixing length model. This is based on the idea that the fluctuating velocity components u' and v' are directly proportional to the mixing length (l_m) times the velocity gradient in that direction [14]. Furthermore, in the same view, the Reynolds shear stress is also proportional to the mixing length and the velocity gradient, the mathematical description of both as shown in Equation 24.

$$\nu_T = l_m^2 \left| \frac{\partial u}{\partial y} \right|$$
$$R_{u'v'} = -\overline{u'v'}\,\rho l_m^2 \left(\frac{\partial u}{\partial y} \right)^2 \tag{24}$$

Where l_m is the mixing length, and is defined as the distance travelled by the fluid particles before blending in with other neighbouring masses or fluid particles. For the given problem statement, taking the velocity gradients in both directions into account, the turbulent eddy viscosity can be mentioned in the form as shown in Equation 25.

$$\nu_T = l_m^2 \sqrt{\left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 + \frac{1}{2} \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)^2} \tag{25}$$

This mixing length is defined based on the physical problem at hand, and is given by empirical relations. For pipe flows, the Nikuradse formulation is often used, which is given by Equation 26.

$$l_m = r_0 \left(0.14 - 0.08 \left(1 - \frac{y}{r_0} \right)^2 - 0.06 \left(1 - \frac{y}{r_0} \right)^4 \right) \tag{26}$$

Where r_0 is the mean radius of the pipe. For the computation of the eddy viscosity at every grid point, the values of the velocity gradients were first computed by a finite difference interpolation, and then the values of the eddy viscosity were calculated at the cell centers by imposing the Neumann boundary condition at all the boundaries, i.e., $\text{grad}(\nu_T) = 0$.

The implementation of the turbulence model is achieved by adding another iteration loop - for the convergence of eddy viscosity similar to the iteration for the determination of pressure.

The implementation of the mixing length turbulence model along with Nikuradse's mixing length formulation is done as highlighted in Code Snippet 5.

```
for i in range(1,nx+1):
  for j in range(1,ny+1):

    #Smagorinsky-Lilly algebraic model equations
    term1_smag = (u[i,j] - u[i-1,j])/h_x
    term2_smag = (v[i,j] - v[i,j-1])/h_y
    term3_smag = 0.5*((u[i-1,j+1] - u[i-1,j-1])/(2*h_y) + (u[i,j+1] - u[i,j-1])/(2*h_y)) \
              + 0.5*((v[i+1,j-1] - v[i-1,j-1])/(2*h_x) + (v[i+1,j] - v[i-1,j])/(2*h_x))

    #Nikuradse's formulation for mixing length
    y = (j-1)*(h_y-w)/ny + w - h_y/2
    term_y = (1-y/w)
    lm = abs(w*(0.14 - 0.08*(term_y**2) - 0.06*(term_y**4)))

    #Calculation of eddy viscosity
    nu_t = ((lm)**2)*np.sqrt(term1_smag**2 + term2_smag**2 + 0.5*(term3_smag)**2)
    mu[i,j] = mu_s + nu_t*rho_s
```

Code Snippet 5: Implementation of the 2D Prandtl mixing length model with Nikuradse's formulation

However, the implementation of the turbulence model was very unstable often leading to non-physical solutions even with very low CFL numbers. Hence, instead of a CFL-criterion-based time stepping, an adaptive time stepping method was devised and implemented.

F. Time-Stepping

A primitive time stepping which relies on finding the largest time step without causing non-physical solutions is implemented to optimise the simulation. Prior to the implementation of this scheme, both the CFL-based time-stepping

scheme and the turbulent-time-scale-based time-stepping scheme were executed and frequent non-physical solutions were obtained. The time-stepping scheme is directly connected to the turbulence model and uses the iterations required for the convergence of dynamic viscosity and the value of the residual of the dynamic viscosity as the driving parameter.

The time-stepping scheme is employed at two parts of the code and control flow - inside the turbulence model iteration loop and outside the turbulence model iteration loop. The implemented time-stepping scheme requires the specification of two user-defined constants or multipliers - δ_1 and δ_2 - the significance of which is discussed in the description of the scheme below.

The modelled turbulence iteration loop is very sensitive to the value of dt, causing an exponential growth in residuals within 10 iterations of the turbulence model loop if the value of dt is unstable. Hence, a simple condition to check the convergence of the turbulence model was to check the number of iterations currently completed. If the loop had exceeded n_1 iterations, the value of dt is multiplied by the multiplier δ_1 and the entire turbulence loop is repeated, deleting the data from all previous iterations of the turbulence loop. The value of δ_1 is typically set in the interval $(0, 1)$ so as to decrease the value of dt for the next iteration. For the current simulation, the value of n_1 was set as 10 and the value of δ_1 to $\frac{2}{3}$.

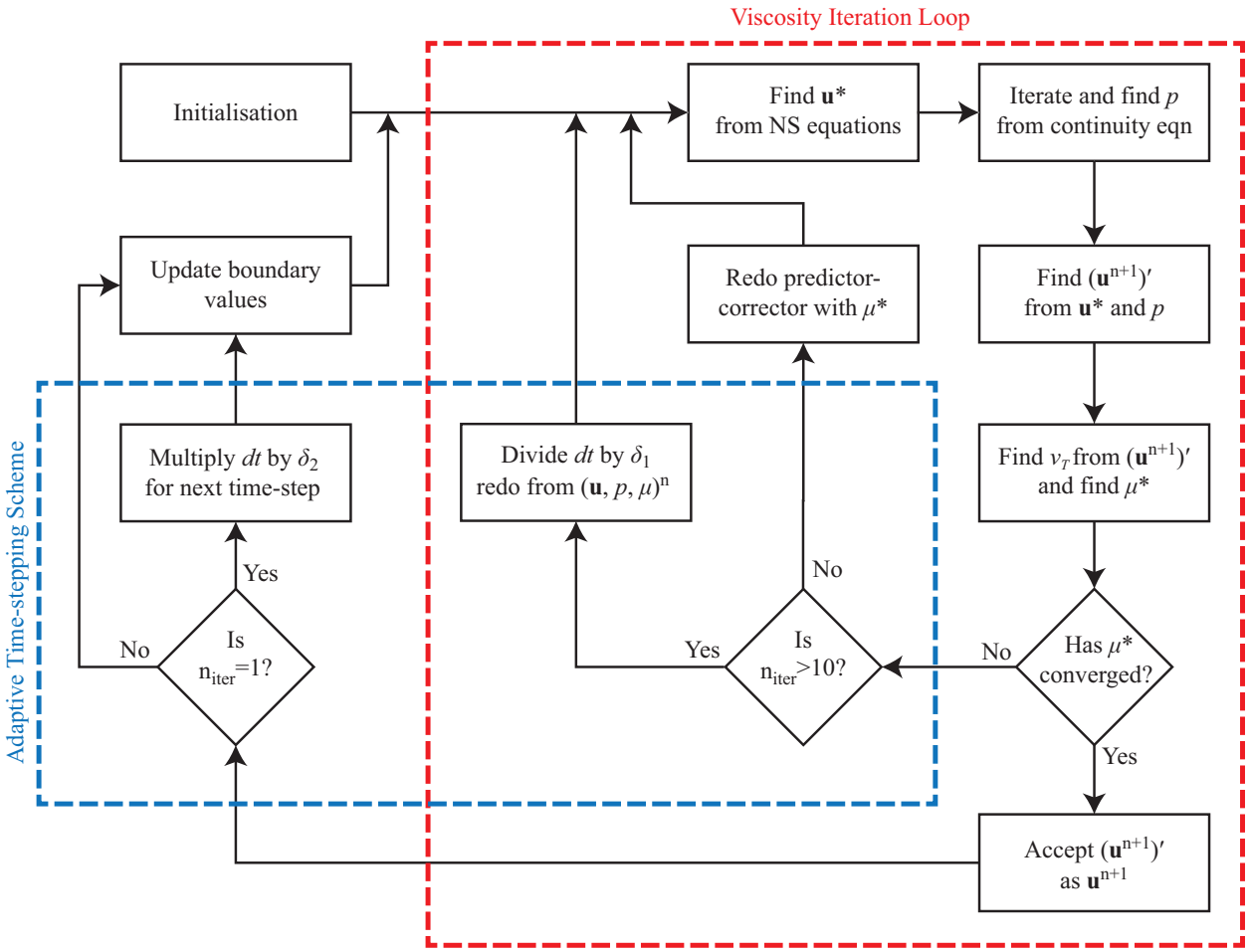

Fig. 5: Control flow for simulation of turbulent pipe flow

If the residuals of the dynamic viscosity reduces to a value below the convergence criteria, it so happens within the chosen limit of 10 iterations as the modelled turbulence model tends to converge rapidly for stable values of dt. On convergence, the determined value of the dynamic viscosity $\mu*$ is accepted as the value of dynamic viscosity at time-step t_{n+1}.

The above time-stepping scheme was found to be extremely stable in operation. However, successive decrease in the

value of dt led to very small orders of magnitude which converged in a single iteration. Hence, a second multiplier step was implemented outside the turbulence iteration loop so as to increase the value of dt if the turbulence model converged too rapidly. Thus, if the turbulence model converged in n_2 iterations, the value of dt for the next time-step was multiplied by δ_2, which is typically in the interval $(0, \infty)$ so as to increase dt. For the current simulation, n_2 was assumed a value of 1 and δ_2 a value of 1.05.

The control flow is depicted by the flowchart given in Figure 5. It is to be noted that $(u^{n+1})'$ in Figure 5 depicts the temporary solution of u^{n+1} and not the fluctuating components of u^{n+1}.

G. Data Handling

Since the iteration process depicted by Figure 5 requires the redo of the predictor-corrector stages in two levels — the pressure-corrector inner iteration, the turbulence model outer iteration —, each loop was paired with a data export subroutine with an additional loop to ensure a smooth resume operation if the program needed to be stopped or the timeout duration for the Python server (mentioned in Section III.D) was reached.

The different levels of data export are as follows:

- Before the start of every time-step, the pressure, velocity and the viscosity matrices are copied to temporary matrices using the `.copy()` command. This is typically done in most Navier-Stokes simulations to ensure that the pressure-corrector algorithm, and subsequently the turbulence loop, has a backup in case the iteration loops do not converge.
- After the successful computation of a time-step, the pressure, velocity and viscosity matrices are saved in CSV format locally. Additionally, the current simulation flow time, the time-step at the previous iteration and the step number are also saved in the CSV format as an additional file locally. This is done to ensure that the simulation may resume smoothly from the last successful time-step when the server timeout was reached.
- Finally, the CSV files are zipped with the step number as the file name of the .ZIP file and saved to Google Drive. This step is vital in transferring the transient profile data to the simulation of the dispersed phase later on.

H. Plots & Figures

1. Velocity Profile

The velocity vectors within the domain at a time $t = 0.5s$ is given in Figure 6.

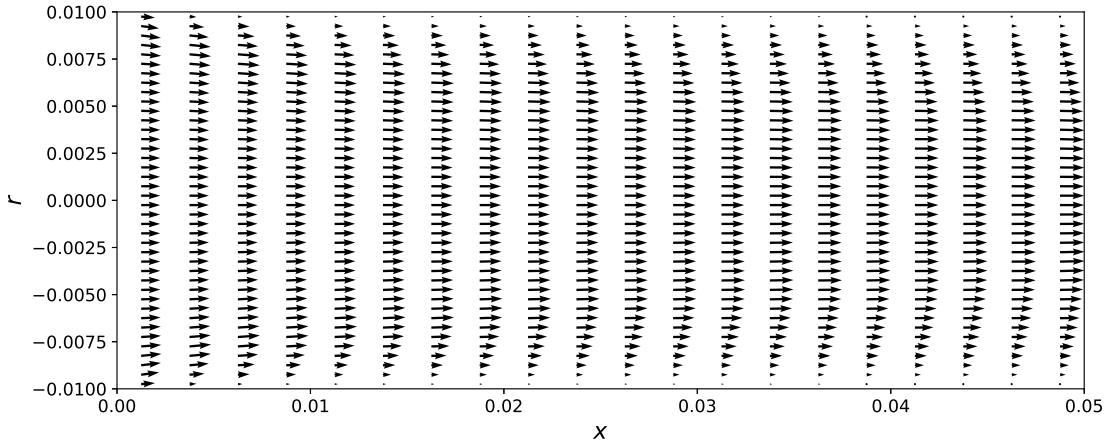

Fig. 6: Velocity vectors for $t = 0.5s$

From Figure 6, it can be observed that the velocity vectors obtained are physically accurate, as the formation of the boundary layer is proved to be physical, which can be deduced from the fact that the velocity gradients in the y-direction are negligible. Therefore, the horizontal velocity vectors remain, which indicates the direction of flow from the inlet to the outlet.

2. X-velocity Profile
The x-velocity profile within the domain at a time $t = 0.5s$ is given in Figure 7.

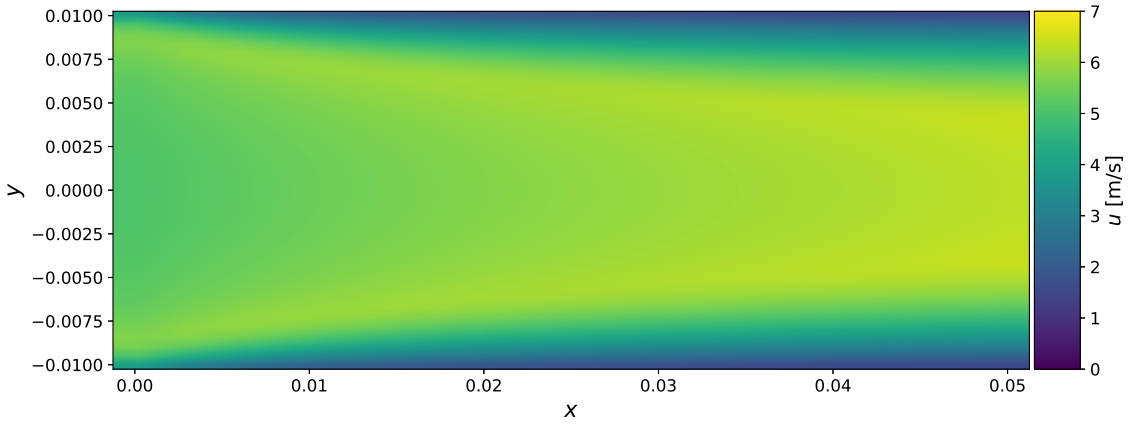

Fig. 7: X-velocity profile for $t = 0.5s$

From Figure 7, it can be seen that the boundary layer has developed according to physical expectations as one proceeds towards the exit of the pipe, which is a good indication that the boundary layer has developed according to the correct flow physics described by the Navier-Stokes equations. However, a more accurate description of the development of the boundary layer is given in Figure 9. Another observation that can be made is the increase in the flow velocity as one proceeds towards the exit of the pipe. This can be accounted for by the fact that the effective cross-sectional area has been decreased because of the increased boundary layer thickness, and since the conservation of mass has to hold true, for the same mass flow rate, the flow velocity has to increase proportionately for a given decrease in cross-sectional area.

3. Y-velocity Profile
The y-velocity profile within the domain at a time $t = 0.5s$ is given in Figure 8.

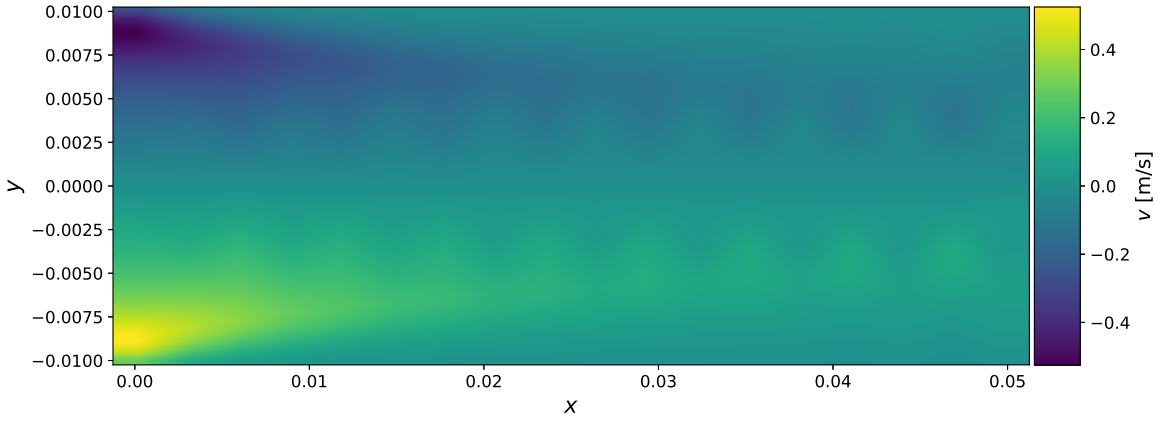

Fig. 8: Y-velocity profile for $t = 0.5s$

From Figure 8, it can be seen that the y-velocity component follows the development of the boundary layer quite accurately, as the velocity gradient in the Y-direction is negligible in comparison to the velocity gradient in the

X-direction. Another observation that can be seen is an almost periodic fluctuation in the y-velocity as the turbulent boundary layer develops. This may be due to the fact that the y-velocity profile experiences a numerical error that propagates downstream in a periodic manner. In particular, this could be caused due to the interpolation errors inherent in the method of interpolation that is used to compute the velocity within a single grid cell for a given time instant.

In addition, there is a large variation in the magnitude of the y-velocity profile in comparison to the x-velocity profile. This can be explained as caused due to the fact that the part of the flow that is closer to the lower wall, where the boundary layer has just begun to develop, has a strong y-velocity gradient. Although this is attributed to a numerical error in terms of incorrectly computing the flow solution near the wall due to the absence of grid points for interpolation, this causes a large error in the values that are calculated for the y-velocity. Therefore, the range of values for the y-velocity is quite large.

Dynamic Viscosity Profile
The viscosity profile within the domain at a time $t = 0.5s$ is given in Figure 9.

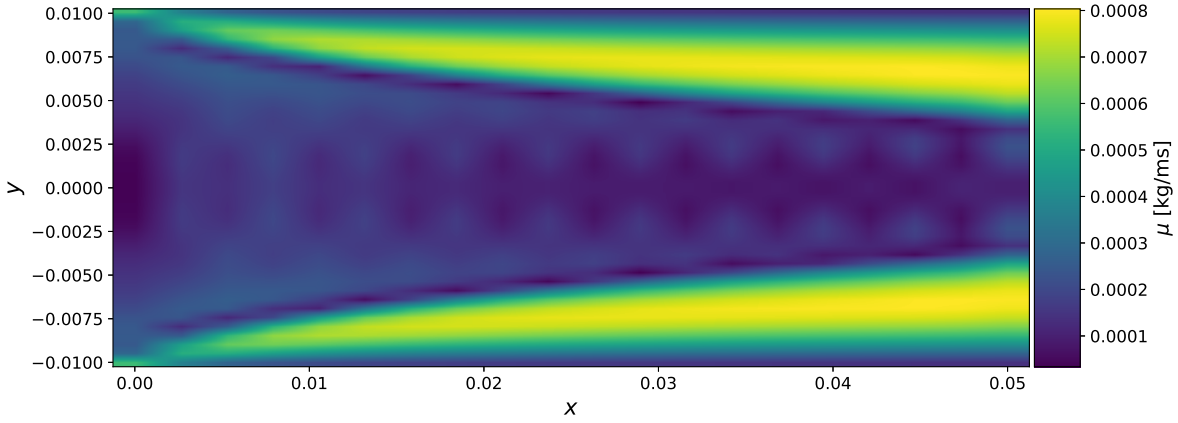

Fig. 9: Dynamic viscosity profile for $t = 0.5s$

As can be observed from Figure 9, the boundary layer development is clearly observed, as the turbulent eddy viscosity becomes gradually higher as the turbulent boundary layer is developed towards the end of the pipe. Another physical reason behind this trend is the increase in mixing between the eddies, which eventually increases the exchange of turbulent kinetic energy. This leads to an increase in the turbulent eddy viscosity ν_T as governed by Equation 25. The numerical error observed in the y-velocity profile can still be observed even in this case.

Another observation that actually highlights one of the shortcomings of the code is the fact that the dynamic viscosity in the part of the boundary layer that is closest to the wall is still zero. This is due to the fact that the current code has not taken into account the various sub-layers within the boundary layer, and in this case, the viscous sub-layer is the part that has not been modelled. As a result of that, the solver simply calculates a zero Reynolds stress tensor in that sub-layer, which results in a zero eddy viscosity in that region. In reality, however, the turbulent eddy viscosity should be the highest in that region.

V. Simulation of the Dispersed Phase

The dispersed phase used in the given problem statement is that of paint, having a density of 1200 kg/m^3. A detailed description of the various steps used to generate the motion of the particles is given in the following sections.

A. Assumptions

In this sub-section, the assumptions are made with respect to the drag force, which is deduced to be the primary force taken into consideration. Hence, the assumptions that are taken into consideration for the given problem statement are

as follows (as adapted from [8]):

- The effect of gravity on the motion of the particles is restricted, as the effect of inertial forces is much more significant than gravitational force. This is because of the fact that the high density of the particles provides a very high inertial force that is of a much higher order of magnitude in comparison to the gravitational force acting on the particle.
- The Basset history force can be neglected in this case, because in comparison to the drag force, for gas-liquid applications, the particle density ρ_p is much larger than the fluid density ρ, which is true in the given problem statement as well. This is because the drag force is dependent on ρ_p and the Basset history force is dependent on ρ, and hence, the result is that the history force is much smaller. In order to provide context, the formulation of the Basset force is given in Equation 27.

$$\mathbf{f}_h = \frac{3}{2} d^2 \rho \sqrt{\pi \nu} \int_{t_0}^{t} \frac{dt'}{(t-t')^{1/2}} \left(\frac{D\mathbf{u}}{Dt'} - \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t'} \right) \tag{27}$$

- In a similar vein as the history force, the added mass and the buoyancy forces can also be neglected, because of their dependence on the fluid density ρ, which can be neglected in comparison to the drag force. Once again, providing a relative comparison between the added mass force and the drag force, given by Equation 28, we can obtain a clearer picture of the importance of the two forces.

$$\frac{\mathbf{f_a}}{\mathbf{f_d}} \simeq \frac{\frac{1}{2}\rho v g}{\frac{3}{4}\rho v (C_d/d) v_t^2} \frac{a_r}{g} = \frac{1}{2} \frac{\rho}{\rho_p} \frac{a_r}{g} \tag{28}$$

  Where v_t is the terminal velocity, which can be taken to be the order of the relative velocity between the particle and the fluid for calculating the drag force. This can be obtained by re-arranging the drag force formulation in Equation 28. Furthermore, making a relative comparison between the history force and the added mass force as shown in Equation 29, we obtain:

$$\frac{\mathbf{f_h}}{\mathbf{f_a}} \simeq 18 \sqrt{\frac{\nu \tau}{d^2}} \tag{29}$$

  Here, the ratio of $\nu\tau$ to d provides an indication of the diffusion length to the particle diameter, which is unity in this case, as $\tau \sim \frac{d^2}{\nu}$. Hence, the added mass force can also be neglected in this case.
- The effect of the Saffman lift force can be neglected, because the lift force is proportional to the fluid density ρ. This can be verified from the formulation of the lift force, given by Equation 30 for a small Re_p.

$$\mathbf{f_l} = C \sqrt{\mu \rho} D^2 |\vec{U} - \vec{V}| \sqrt{\frac{\mathrm{d}u}{\mathrm{d}y}} \tag{30}$$

  Furthermore, since the flow vorticity is much larger than the particle size, the drag force is still more dominant in comparison to the lift force. Therefore, the lift force
- Since the volume fraction of the continuous phase is much larger than the volume fraction of the dispersed phase, the particles can be considered to have a large mean free path, as has been described in Section III.C. Therefore, the effect of particle collisions can be neglected.
- The reflections of the particles from the wall are considered to be elastic in nature, as the analysis of the motion of particles would be easier for unsteady simulations like the given problem statement.
- The drag force acting on the particles is assumed to follow the drag law given by Schiller and Neumann [15] in Equation 34. This is because the particle Reynolds number is lesser than 1000, and hence, this formulation for C_d is valid.
- While calculating the fluctuating velocity components by using the Monte-Carlo simulation method in Section III.C, the assumption of isotropic turbulence is extremely important to be taken into account, as it simplifies the calculations in all directions.

B. Discretization of Equations of Motion

The general equation of motion of the particle that is immersed in a fluid, in the continuous form, is given by Equation 31 [8].

$$m_p \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = m_p \mathbf{g} - \rho v \left(\frac{D\mathbf{u}}{Dt} - \mathbf{g} \right) + \mathbf{f}_d + \mathbf{f}_l + \mathbf{f}_a + \mathbf{f}_h + \mathbf{f}_{add} \tag{31}$$

Where $\mathbf{f}_l$, $\mathbf{f}_d$, $\mathbf{f}_a$, $\mathbf{f}_h$ and $\mathbf{f}_{add}$ denote the lift, drag, added mass, history force and the additional forces such as electrostatic and magnetic forces between particles respectively. From Section V.A, it is clear that the major forces acting on the particle for the given problem statement are the inertial forces as well as the drag force. Therefore, the simplified form of Equation 31 is given by Equation 32.

$$\rho_p \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \rho_p \mathbf{g} - \rho_p \frac{\vec{\mathbf{v}}^n - \vec{\mathbf{u}}^n}{\tau_p} \tag{32}$$

The drag force can be modelled in multiple ways, with the general formulation given by Equation 33.

$$\mathbf{f}_d = -\frac{3}{4} \rho v \frac{C_d}{d} |\mathbf{v} - \mathbf{u}| (\mathbf{v} - \mathbf{u}) \tag{33}$$

If the empirical relation provided by Schiller and Naumann [15] is used, then the drag force can be expressed as given in Equation 34.

$$\mathbf{f}_d = 1 + 0.15 {Re_p^n}^{0.687} \tag{34}$$

Where Re_p is the particle Reynolds number, which is given by Equation 35.

$$Re_p = \frac{\rho_p |\vec{v}^n - \vec{u}^n| D_p}{\mu} \tag{35}$$

The corresponding particle relaxation time, τ_p, is given by Equation 36.

$$\tau_p^n = \frac{\rho_p}{18\,\mu} \frac{D_p^2}{(1 + 0.15\,(Re_p^n)^{0.687}))} \tag{36}$$

Where f_d is an empirical drag coefficient, that is used to imply the dependence of Reynolds Number on the body forces. It is to be noted that Equation 32 is a simplification of the BBO equation, taking into account the assumptions given in Section V.A.

The discretization of Equation 32 is performed by using a Forward-Euler finite difference scheme for the time derivative, and the final form of the governing equation is given by Equation 37.

$$\frac{\vec{v}^{n+1} - \vec{v}^n}{\Delta t} = -\frac{\vec{v}^n - \vec{u}^n}{\tau_p} \tag{37}$$

C. Discrete Eddy Simulation

The Discrete Eddy Simulation concept is based on the assumption of isotropic turbulence [16]. In this method, the instantaneous velocity of the eddies is obtained from the randomized velocity distribution given by a Gaussian profile. This can be viewed as being constituted by a mean velocity component and a fluctuating velocity component, which can be taken to be an R.M.S. fluctuation (given by σ). If k is the turbulent kinetic energy of the flow, then the fluctuating component of velocity can be obtained by using the relation given by Equation 38.

$$\sigma = \sqrt{\frac{2}{3} k} \tag{38}$$

Where $\sigma = \sqrt{\overline{u'^2}} = \sqrt{\overline{v'^2}} = \sqrt{\overline{w'^2}}$, assuming isotropic turbulence. This instantaneous velocity is obtained by assuming that the particle motion is influenced by the fluid for a certain period of time. A turbulent eddy is further characterized by a length scale, which is defined by Equation 39. Furthermore, the interaction between the eddies is also taken into account by the eddy life-time as well as the eddy turnover time, both of which are defined as given by Equation 41.

$$l = C_\mu \frac{k^{1.5}}{\epsilon} \tag{39}$$

$$T_e = \frac{l}{\sqrt{\frac{2k}{3}}} \tag{40}$$

$$T_t = \frac{l}{|\overline{U} - \overline{U_p}|} \tag{41}$$

where k is the turbulent kinetic energy of the flow, $\overline{U}$ is the mean velocity of the fluid, and $\overline{U_p}$ is the mean velocity of the particle at a given instant of time. The interaction time is then defined as the minimum of the eddy lifetime and the eddy turnover time. It is defined as the time taken for interaction between the eddy and the particle, defined by Equation 42.

$$T_{\text{inter}} = min(T_e, T_t) \tag{42}$$

This time is added to the flow time and is updated at every time instant in order to ensure that the particle motion is influenced by the correct flow physics, defined by the interaction time. The implementation of the discrete eddy simulation is shown in Figure 10.

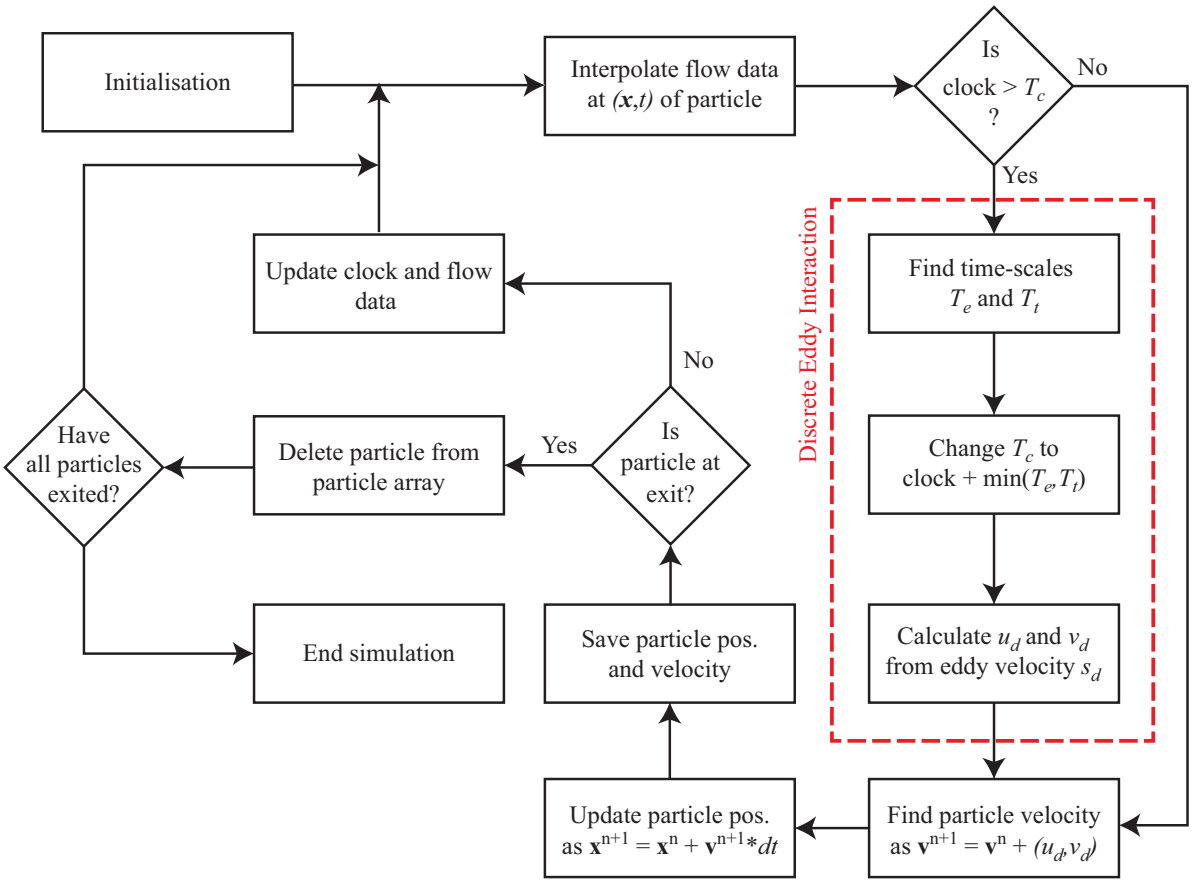

Fig. 10: Control flow for discrete eddy simulation of particles

D. Importing Continuous Phase Data

The phase data for the continuous phase is imported by reading the zip files in which the velocity components (x- and y-directional velocity components u and v respectively), the pressure (p), the dynamic viscosity (μ) and the time are stored. This is done by using the zip file reading module `ZipFile` in Python. The code that is used to import the continuous phase data is shown in Code Snippet 6. The `pandas` module, along with `numpy`, is used to read data from the CSV file and convert it into numerical arrays.

```
# Location of the fluid-phase data
string = "/content/drive/My Drive/Computational Multiphase Flow/%d/%d.zip"

# Load velocity and viscosity values omitting ghost cells
zfile = ZipFile(string %(int(i/1000),i), 'r')
ut = np.array(pd.read_csv(zfile.open('u.csv'), sep=',', header=None))[0:-1,1:-1]
vt = np.array(pd.read_csv(zfile.open('v.csv'), sep=',', header=None))[1:-1,:]
mut = np.array(pd.read_csv(zfile.open('m.csv'), sep=',', header=None))[:,1:]

# Get the time-step for the current step
dt = np.array(pd.read_csv(zfile.open('t.csv'), sep=',', header=None))[0,0]
```

Code Snippet 6: Importing zip files from the continuous phase

The continuous phase is simulated in an axisymmetric fashion, with a symmetry boundary condition at the bottom wall. Since the motion of the particles is not perfectly axisymmetric, courtesy of the randomness introduced by the discrete eddy model, the simulation of the particulate phase has to be carried out in full Cartesian form. Hence, the axisymmetric fluid side data is converted to Cartesian form using code given in Code Snippet 7.

```
# Copy velocity and viscosity arrays on -y axis and append
ug = np.vstack((-1*ut.T[-1],np.flipud(ut.T),ut.T,-1*ut.T[-1])).T

vg = np.vstack((-1*np.flipud(vt.T),vt.T))
vg = np.delete(vg,len(vt),0).T
vg = np.vstack((-1*vg[0],vg,vg[-1]))

mug = np.vstack((np.flipud(mut.T),mut.T)).T
```

Code Snippet 7: Converting axisymmetric fluid-side data to Cartesian form

E. Handling Continuous Phase Data

In order to couple the particle motion with the fluid profile, two user-defined functions are used to provide the input from the continuous phase to the dispersed phase. This is done using two different functions, namely the `index_finder` and the `interpolate`. The functionalities of these are explained as follows:

- The `index_finder` function is used to obtain the fluid grid location at the position of the particle. Since the particle positions are calculated on the particle grid which is faux-continuous with a precision of the float data type, the velocity components are transposed on the fluid grid which is discretised. This is also done for the dynamic viscosity values, as they are also stored at different locations of the fluid grid in comparison to the particle grid. The output of this function are the velocity components in each direction, as well as the dynamic viscosity in each direction, and is shown in Code Snippet 8.
- The `interpolate` function is used to interpolate the values of velocity and viscosity from the fluid phase data at the position of the particle, considering the values of velocity and viscosity from the nearby fluid phase nodes. The interpolation algorithm used for this purpose is the bilinear interpolation. Since the fluid phase data has a staggered grid arrangement with a different grid for horizontal and vertical components of fluid velocity and a separate cell-centered grid for fluid viscosity, the interpolation for each value is done separately. This is depicted in Code Snippet 9.

```
    def index_finder(px,py):

      # find index of nearby node in the staggered grid for u,v,mu
      ux = int((px)/h_x)
      uy = int((py+w+h_y/2)/h_y)
      vx = int((px+h_x/2)/h_x)
      vy = int((py+w)/h_y)
      mux = int((px+h_x/2)/h_x)
```

```
    muy = int((py+w+h_y/2)/h_y)

    # near wall treatment specific to domain
    if ux==20:
      ux = 19
    if vy==40:
      vy = 39

  return [ux, uy, vx, vy, mux, muy]
```

Code Snippet 8: Index finding algorithm

```
  def interpolate(px,py,ug,vg,mug):

      # interpolation for horizontal fluid velocity component
      c1 = px - uxgrid[ux,uy]
      c2 = h_x - c1
      c3 = py - uygrid[ux,uy]
      c4 = h_y - c3
      uf = (c2*c4*ug[ux,uy] + c4*c1*ug[ux+1,uy] + c3*c2*ug[ux,uy+1] +
      c1*c3*ug[ux+1,uy+1])/(h_x*h_y)

      # interpolation for vertical fluid velocity component
      c1 = px - vxgrid[vx,vy]
      c2 = h_x - c1
      c3 = py - vygrid[vx,vy]
      c4 = h_y - c3
      vf = (c2*c4*vg[vx,vy] + c4*c1*vg[vx+1,vy] + c3*c2*vg[vx,vy+1] +
      c1*c3*vg[vx+1,vy+1])/(h_x*h_y)

      #interpolation for fluid viscosity
      c1 = px - muxgrid[mux,muy]
      c2 = h_x - c1
      c3 = py - muygrid[mux,muy]
      c4 = h_y - c3
      muf = (c2*c4*mug[mux,muy] + c4*c1*mug[mux+1,muy] + c3*c2*mug[mux,muy+1] +
      c1*c3*mug[mux+1,muy+1])/(h_x*h_y)

  return [uf,vf,abs(muf)]
```

Code Snippet 9: Interpolation function definition

F. Particle Setup

The particles are defined by a separate class with its physical and flow parameters as its properties as given in Code Snippet 10. These physical parameters include the particle density and the particle position. The flow parameters include particle displacements and the velocity components in each direction. Additionally, the drift velocity calculated from the interaction of the particle with a discrete eddy is stored individually. Hence, every discrete eddy is specific and unique to every particle. The class definition is also defined with attribute functions `move(un,vn,dt)` and `reflect()` to update the particle position and velocity after each time-step and check for wall interactions.

```
class particles:
  def __init__(self, u, v):

    # mean diameter for paint particle
    self.diameter = abs(random.normal(loc=30e-6, scale=5e-6))
```

```
    # particle is initially at the inlet and distributed normally around the center with S.D 0.5mm
    self.x = 0 # particle is initially at the inlet
    self.y = random.normal(loc=0,scale=5e-4) #2*w*(random.random() - 0.5)

    # initial velocity for the particle
    self.u = u
    self.v = v

    # mean density of paint
    self.rho_p = 1200

    # store initial y for post-proc
    self.y0 = self.y

    # variables for discrete eddy interaction
    self.condition = 0
    self.sd = 0
    self.ud = 0
    self.vd = 0

  def move(self,un,vn,dt):

    # time integration to find position
    self.x = self.x + (self.u + un)*dt/2
    self.y = self.y + (self.v + vn)*dt/2
    self.u = un
    self.v = vn

    #reflect if necessary
    self.reflect()

  def reflect(self):

    # elastic collision if particle hits the upper and lower walls
    if self.y>0.01:
      bound = 0.01
      self.y = bound - (self.y-bound)
      self.v = -self.v

    if self.y<-0.01:
      bound = -0.01
      self.y = bound + (bound - self.y)
      self.v = -self.v
```

Code Snippet 10: Particle class setup

The particle phase motion is built into the class as the functions `move(un,vn,dt)` and `reflect()`. The velocity of the particle is calculated dynamically and passed as arguments to the function `move(un,vn,dt)` and a simple Newtonian displacement equation is used to determine the new location of the particle. After the new location is determined, it is checked whether the particle is within the fluid domain. If the particle has crossed a rigid wall, a simple specular reflection algorithm is implemented to redirect the particle back into the domain.

G. Single Particle Simulation

The single particle simulation was used to as a foundation to test if the particle simulation was working properly. The basic algorithm for this is provided in the following steps:

- The particle velocities are interpolated based on their individual locations. This is done for both the x and y velocity components.

- The particle Reynolds number is then defined on the basis of Equation 35, which is in turn, defined on the basis of the relative velocity between the particle and the fluid.
- The particle relaxation time is then defined on the basis of Equation 36. This is then used to store the particle velocities in an array for each time step.
- As long as the maximum flow time is not exceeded, the loop runs by defining the particle crossing time and the interaction time based on Equation 42. The time is regularly updated by using this interaction time as a counter.
- As a final step, the particle velocity components of the eddy interaction are randomized by using a normal distribution function and updating the particle velocity for each time step.

The code written for a single particle is given in Figure 11.

```
# initialise a particle
p = particles(up,vp)
x0 = p.x
y0 = p.y

# string for fluid phase data import
string = "/content/drive/My Drive/Computational Multiphase Flow/%d/%d.zip"

# initialise discrete eddy interaction time
condition = 0

# run for 1000 time steps
for i in range(1,1000):

  # import fluid data
  zfile = ZipFile(string %(int(i/1000),i), 'r')
  ut = np.array(pd.read_csv(zfile.open('u.csv'), sep=',', header=None))[0:-1,1:-1]
  vt = np.array(pd.read_csv(zfile.open('v.csv'), sep=',', header=None))[1:-1,:]
  mut = np.array(pd.read_csv(zfile.open('m.csv'), sep=',', header=None))[:,1:]
  dt = np.array(pd.read_csv(zfile.open('t.csv'), sep=',', header=None))[0,0]
  clock = np.array(pd.read_csv(zfile.open('t.csv'), sep=',', header=None))[1,0]

  # convert from axisymmetric to full cartesian
  ug = np.vstack((-1*ut.T[-1],np.flipud(ut.T),ut.T,-1*ut.T[-1])).T
  vg = np.vstack((-1*np.flipud(vt.T),vt.T))
  vg = np.delete(vg,len(vt),0).T
  vg = np.vstack((-1*vg[0],vg,vg[-1]))
  mug = np.vstack((np.flipud(mut.T),mut.T)).T

  # interpolate fluid data at particle position
  [uf, vf, muf] = interpolate(p.x,p.y)

  # find velocity difference between particle and fluid
  Delta_v = np.array([p.u - uf, p.v - vf])

  # discrete eddy interaction
  if clock > p.condition:
    term_y = 1-(w-np.abs(p.y))/w
    lm = abs(w*(0.14 - 0.08*(term_y**2) - 0.06*(term_y**4))) #prandtl length scale

    T_t = lm / np.linalg.norm(Delta_v) #eddy turnover time
    T_e = (lm**2)*rho_f/(muf-1e-5) #eddy lifetime

    p.condition = clock + min(T_c,T_t) #update eddy interaction time
    p.sd = abs((muf - 1e-5)/(rho_f*lm)) #get eddy velocity

    p.ud = np.random.normal(loc=uf, scale = p.sd) - uf #randomise direction
    p.vd = np.random.normal(loc=vf, scale = p.sd) - vf
```

```
# append eddy velocity to interpolated fluid velocity
uf = uf + p.ud
vf = vf + p.vd

Delta_v = np.array([p.u - uf, p.v - vf])
Re_p = p.rho_p * np.linalg.norm(Delta_v) * p.diameter / (muf) # Reynolds number
T_p = p.rho_p*(p.diameter)**2 / (18 * (muf) * (1 + 0.15*(Re_p**0.687))) # relaxation time

# find new particle velocity
[un,vn] = np.array([p.u,p.v]) - dt*Delta_v/T_p

# update particle position
p.move(un,vn,dt)
```

Code Snippet 11: Single particle simulation

H. Multiple Particle Simulation

Following the execution of the simulation of a single particle, the code was altered to simulate multiple particles simultaneously. The particles are initialised at the inlet with a normal distribution for its vertical position, with mean at the center of the channel and standard deviation of a quarter width of the channel. The particles are simulated individually at each time-step and when they reach the exit of the domain, they are removed from the simulation loop and the exit position and velocity data were stored for post-processing. The simulation was coded to end when all the particles exited the domain.

Additionally, for storing the particle data as it exited the domain, an array was pre-initialised. A function named `storeit` was created to ease the export of particle data before deletion. The function is given in Code Snippet 12.

```
  storage = []
  storage.append(["Diameter", "y0", "y", "u", "v", "Clock"])
  storage = np.array(storage)

  def storeit(storage,stuff):
    stuff = np.array(stuff)
    storage = np.vstack((storage,stuff))
    return storage
```

Code Snippet 12: Particle position definition

For multiple particles, the major difference lies in the fact that the particle velocities at each spatial location are taken into consideration for interpolating the particle velocities u and v as shown in Code Snippet 13. There is an additional condition that the particles would be removed once they are out of the domain of the pipe.

```
#initialise particle array with 1000 particles
p_array = []
num = 1000
for i in range(0,num):
  p_array.append(particles(up,vp))

# simulate until all particles leave domain
i = 0
while len(p_array)>0:
  i = i+1

  # import fluid data
  zfile = ZipFile(string %(int(i/1000),i), 'r')
  ut = np.array(pd.read_csv(zfile.open('u.csv'), sep=',', header=None))[0:-1,1:-1]
```

```
vt = np.array(pd.read_csv(zfile.open('v.csv'), sep=',', header=None))[1:-1,:]
mut = np.array(pd.read_csv(zfile.open('m.csv'), sep=',', header=None))[:,1:]
dt = np.array(pd.read_csv(zfile.open('t.csv'), sep=',', header=None))[0,0]
clock = np.array(pd.read_csv(zfile.open('t.csv'), sep=',', header=None))[1,0]

#  convert from axisymmetric to full cartesian
ug = np.vstack((-1*ut.T[-1],np.flipud(ut.T),ut.T,-1*ut.T[-1])).T
vg = np.vstack((-1*np.flipud(vt.T),vt.T))
vg = np.delete(vg,len(vt),0).T
vg = np.vstack((-1*vg[0],vg,vg[-1]))
mug = np.vstack((np.flipud(mut.T),mut.T)).T

# loop over all particles
j = 0
while j<len(p_array):

    p = p_array[j]

    # interpolate fluid data at particle position
    [uf, vf, muf] = interpolate(p.x,p.y,ug,vg,mug)

    # find velocity difference between particle and fluid
    Delta_v = np.array([p.u - uf, p.v - vf])

    # discrete eddy interaction
    if clock > p.condition:
      term_y = 1-(w-np.abs(p.y))/w
      lm = abs(w*(0.14 - 0.08*(term_y**2) - 0.06*(term_y**4))) #prandtl length scale

      T_t = lm / np.linalg.norm(Delta_v) #eddy turnover time
      T_e = (lm**2)*rho_f/(muf-1e-5) #eddy lifetime

      p.condition = clock + min(T_c,T_t) #update eddy interaction time
      p.sd = abs((muf - 1e-5)/(rho_f*lm)) #get eddy velocity

      p.ud = np.random.normal(loc=uf, scale = p.sd) - uf #randomise direction
      p.vd = np.random.normal(loc=vf, scale = p.sd) - vf

    # append eddy velocity to interpolated fluid velocity
    uf = uf + p.ud
    vf = vf + p.vd

    Delta_v = np.array([p.u - uf, p.v - vf])
    Re_p = p.rho_p * np.linalg.norm(Delta_v) * p.diameter / (muf) #Reynolds number
    T_p = p.rho_p*(p.diameter)**2 / (18 * (muf) * (1 + 0.15*(Re_p**0.687))) #relaxation time

    # find new particle velocity
    [un,vn] = np.array([p.u,p.v]) - dt*Delta_v/T_p

    # update particle position
    p.move(un,vn,dt)

    # save in particle array
    p_array[j] = p

    # if particle reaches outlet, save particle data in storage array and delete
    if p.x > 0.05:
      stuff = [p.diameter, p.y0, p.y, p.u, p.v, clock]
```

```
        storage = storeit(storage,stuff)
        del p_array[j]
        num = num-1
        print("A particle reached the end.")
        j = j-1

      j = j + 1

  # if all particles exited domain, save the stored particle data as csv and exit
  if len(p_array) == 0:
    np.savetxt("data.csv",storage,fmt="%s",delimiter=",")
    break
```

Code Snippet 13: Multiple particle simulation

VI. Results and Discussion

The following section discusses the results obtained from the simulation of the particles in the multiphase flow regime.

- Visualisation of particle motion
- Variation of particle diameter at the pipe exit.
- Variation of particle diameter at the pipe exit.
- Variation of exit velocity at the exit
- Histogram of particles.
- Variation of initial positions at pipe exit.
- Variation of velocity at pipe exit.

A. Visualisation of particle motion

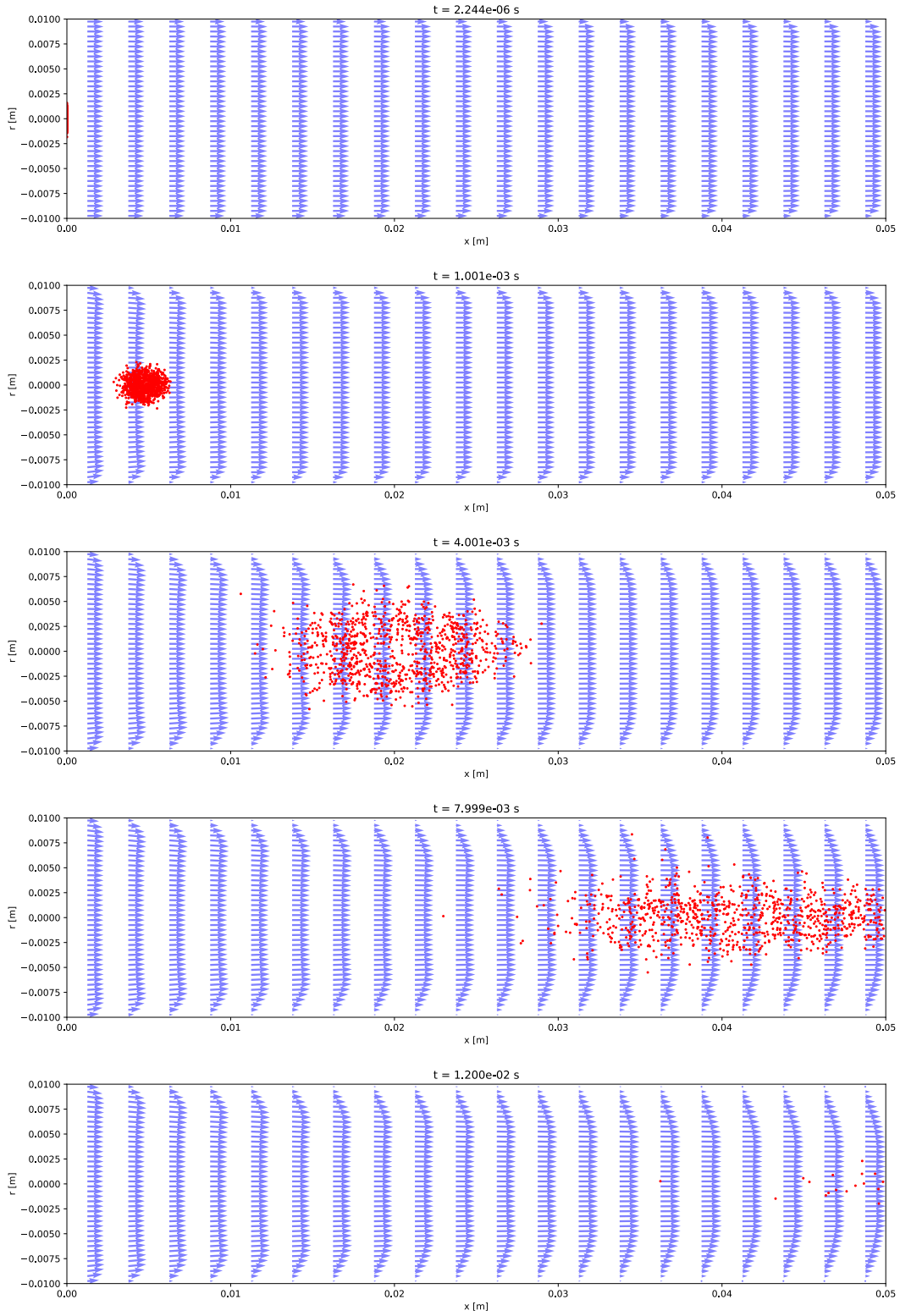

Fig. 11: Visualisation of particle motion through channel, $t = \{0, 1, 4, 8, 12\}$ ms; Particles in red, flow profile in light blue.

The motion of the particles and the development of the flow profile with time can be seen in Figure 11. A video of the same is available in the link `https://github.com/shyam97/cmf/blob/Checkpoint-Delta/cmf_1.mp4?raw=true`.

B. Variation of particle position at pipe exit

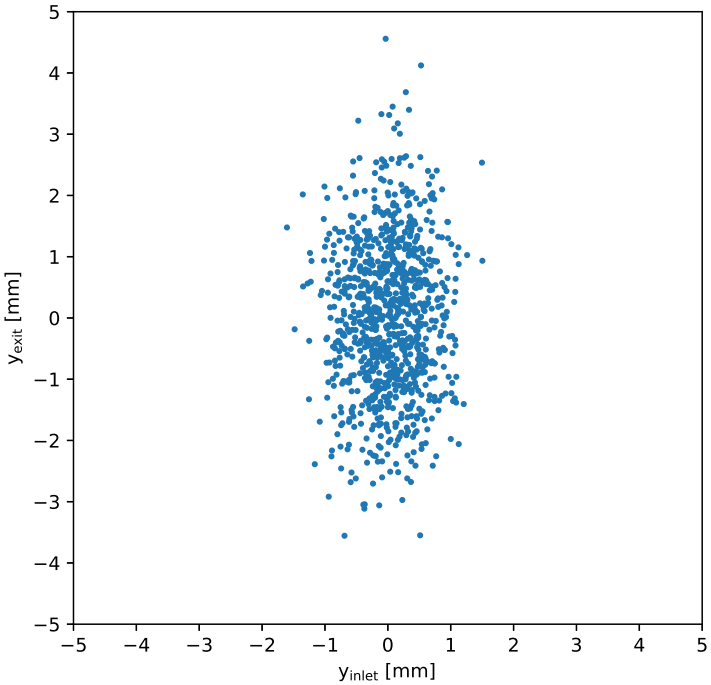

Fig. 12: Particle position at pipe exit vs. pipe inlet

The variation of the particle position in Figure 12 shows some interesting features of the pipe flow. First, it can be observed that the distribution of the particles follows an even distribution around the $y_{inlet} = 0$ line. This implies that at the pipe exit, the particles are symmetrically distributed around the centerline. This is a consequence of the symmetry inlet velocity condition, which was maintained throughout the pipe flow. Second, the particle positions are distributed between -4 and 5 mm, with most of the particles being concentrated in the region around the $y_{exit} = 0$ line. Also, this can be seen along the $y_{inlet} = 0$ line as well. This indicates that the particles are symmetrically distributed along both the horizontal as well as vertical directions respectively.

Further, another observation that can be made is that the distribution of the particle positions at the pipe exit is also reminiscent of a normal distribution, with most of the particles being distributed around a 'mean' position $y_{exit} = 0$ and a variance of roughly 1 mm, corresponding to which the particle distribution changes. And in accordance with the theory, roughly 99% of the particles can be found in the range of -3 mm to +3 mm of the pipe exit. This confirms the Gaussian-like distribution of the particles. This distribution is also but expected, given the fact that the initial introduction of particles was given as a normal distribution at the inlet, and given that the velocities also followed a similar trend, the final particle positions also reflected this. Therefore, the distribution obtained is physical and justified.

C. Histogram of particles at inlet and exit

From Figure 13, it can be observed that the standard deviation of the vertical position of the particle (denoted by σ in the histogram) is higher at the exit than at the inlet. This follows the discussion given in VI.B. As stated in VI.B, the position at the exit conforms to normal distribution similar to the inlet. The increase in standard deviation from $\sigma = 0.51$ mm to $\sigma = 1.26$ mm can be attributed to the randomisation of velocity of the particles by discrete eddies. This hypothesis is also reinforced by the fact that the horizontal velocity component of flow (as seen in Figure 7) is an order of magnitude

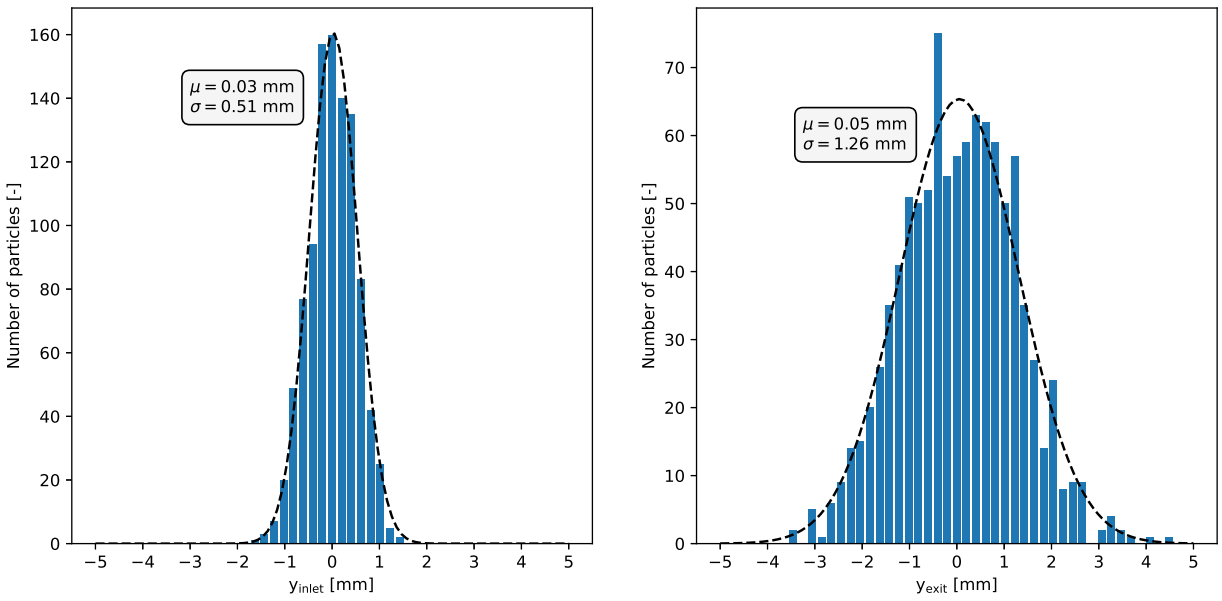

Fig. 13: Histogram depicting the variation of the particle exit diameters for different mean values; left - position at inlet, right - position at exit

stronger than the vertical velocity component of flow (as seen in Figure 6). Hence, the distribution along the vertical axis, if noticeable, must have come from the eddy component of velocity which is randomised in all directions.

D. Variation of exit velocity

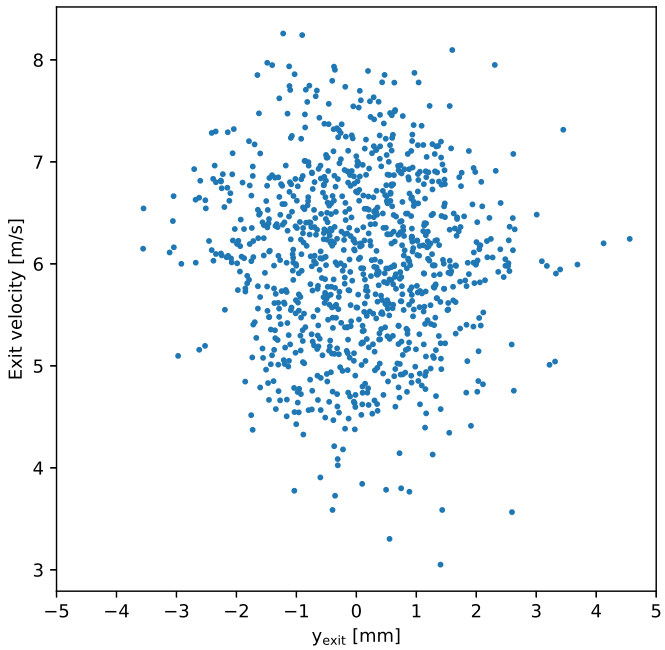

Fig. 14: Exit velocity of particles vs. exit position at pipe exit

From Figure 14, it can be observed that unlike the distributions of the particles at various exit times and positions, the particle exit velocity has a much greater variation with respect to the exit position. This can be attributed to the fact unlike the particle positions that are diameter dependent and based on a normal distribution with the initial position, the particle velocities are directly obtained as a bilinear interpolation from flow velocities and the randomised eddy velocity, which is not distributed normally. Therefore, there is no strict constraint for the particles to be distributed in a normal fashion with respect to the position.

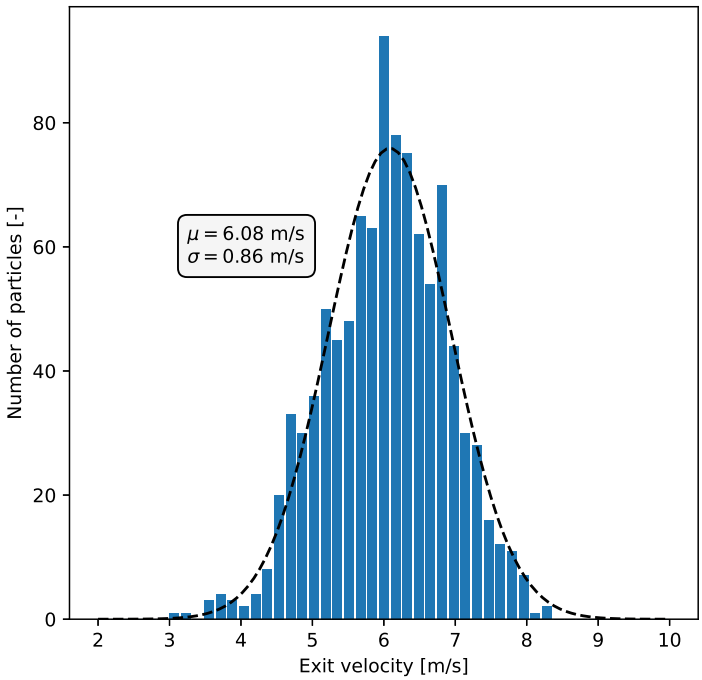

Fig. 15: Histogram of particle exit velocity

Furthermore, from Figure 15, the exit velocity is also seen to conform to a normal distribution. However, this is expected because the mean velocity on the particles is directly related to the mean velocity of flow, which is approximately 7 m/s (as seen in Figure 7). The deficit of 1 m/s may be attributed to the force on the particles due to the difference in density relative to the fluid.

E. Variation of particle exit times

From Figure 16, it can be observed that the particle exit times for different particle diameters follows a similar trend as the particle positions. To this end, the particle distribution at various time instances reflected some interesting characteristics. First, it is interesting to note that the particles having a diameter in this range also have exit times of a very similar magnitude (between 7.5 and 12.5 ms). This can be because of the fact that the body forces acting on the particles and viscous effects combine to provide a more or less uniform effect on all the particles involved. Also, given that the boundary layer has developed quite well at the pipe exit, which can be seen from Figure 6. Therefore, the statement of a uniform body force acting on the particles is justified.

Figure 17 shows the distribution in particle exit times. The mean exit time of μ = 9.35 ms corresponds to the expected mean time for a particle to cross the domain with a mean velocity of 6 m/s. Furthermore, the distribution does not accurately conform to a normal distribution since there is a slight skew in the distribution favoring higher exit times. This can be seen in the drastic drop in frequency for lower exit times versus the more gradual drop at higher exit times.

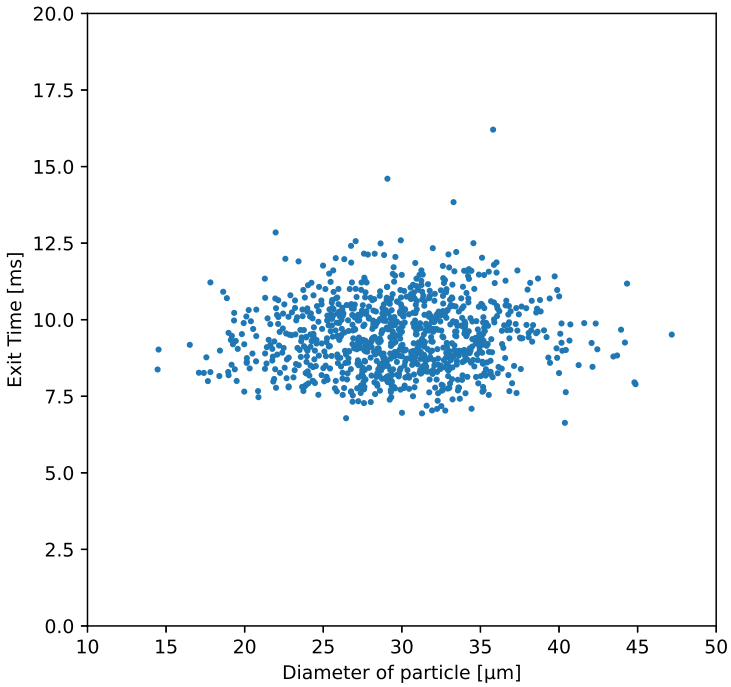

Fig. 16: Particle exit time vs. Diameter of particle

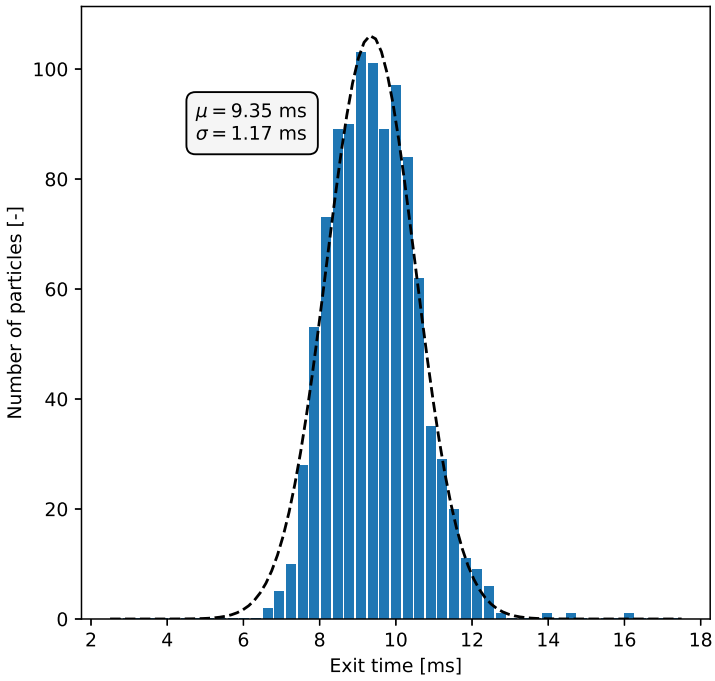

Fig. 17: Histogram of particle exit times

F. Variation of velocity at pipe exit

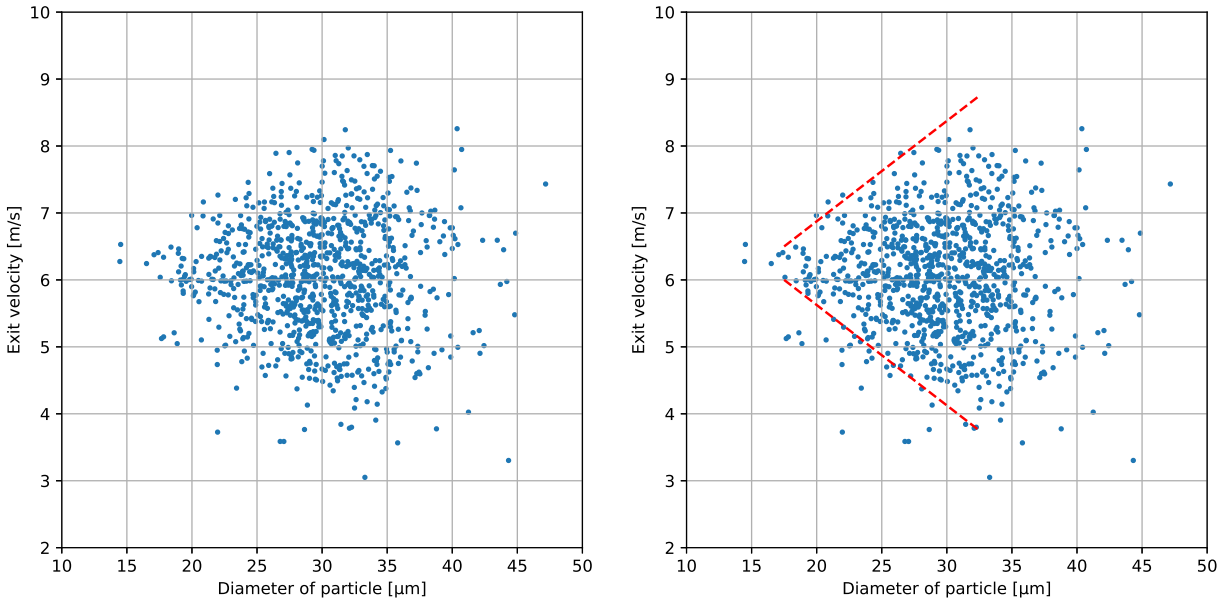

Fig. 18: Exit velocity of particles vs. Diameter of particles; left - raw data, right - data with linear construction depicting limit of variation

The variation in the exit velocity of the particles versus their diameter is shown in Figure 18. The distribution is random upon first look but a linear trend can be visualised as in the subplot to the right. The construction shows the approximate line showing the limit of variation in exit velocity with respect to the diameter of the particle. This indicates that the variation in the exit velocity can be linearly correlated to the particle diameter, meaning particles of smaller size are less likely to deviate from the mean exit velocity. This is on par with the theory of particulate flow which states that the particle relaxation time is smaller for smaller particles. Hence, in this case, particles of smaller size conform to the mean flow faster and are not affected by the eddies. Whereas, particles of larger size have a much wider spread in the exit velocities indicating that their motion trajectory is dependent on the eddies more than their smaller counterparts.

VII. Conclusion

This report covers the description of a code that provides a simplified setup involving the motion of particles through a spray pipe that is cylindrical in nature, but have a small width. The simulation included the effects of turbulence as well as viscous dispersion. The simulation setup consists of two sections, namely the fluid side and the solid side separately. The fluid side, or the continuous phase, is based on the Prandtl mixing length model, incorporating the Nikuradse's formulation of the variation of the mixing length with the wall-normal distance. The solid side, or the dispersed phase, is based on the various kinds of particle motions that can be expected in multiphase flows, such as particle reflection off walls (specular reflection), deleting particles at the end of the wall in order to ensure that there is no accumulation after each time step and so on. The logic behind the calculation of the velocities of each particle in space is also obtained by using a bilinear interpolation, which is introduced as a separate module. There are also additional modules on the variation of the location of the particle with time, as well as separate classes for generating particles and their corresponding motions. This modularity of the code also provides a large scope for the improvement of the existing code. This can be done by the addition of more modules that describe the physics of the problem at hand more accurately, such as inelastic collisions and the coalescence of particles. The corresponding theory behind both the fluid and the solid sides is described in some detail for each section respectively. The results obtained from the code are then discussed in some detail, with emphasis laid on the trends of the x and y velocity components, as well as the eddy viscosity variations in the domain. There are also comments made on the distribution of the particle position and the velocity at the pipe outlet.

VIII. Scope for Improvement

Although this problem statement provides an initial starting point for the purpose of observing the basic phenomenon behind pipe flows, there are many simplifications that have been used, which have been clearly explained in Section III. Therefore, keeping these assumptions in mind, there are certain points that can be improved upon in the code, in order to obtain a more realistic flow solution for the purpose of future analysis. These include the following:

1) The current code makes an assumption that the reflections of the particles from the wall are all elastic in nature. Therefore, by introducing a restitution coefficient that is dependent on the properties of the wall, *inelastic collisions* between the particles and the wall can be introduced into the picture. This would be more realistic, as there is always a certain amount of kinetic energy that is lost after the collision of the particle with the wall, which is not assumed in the current code.
2) In addition to the first point, the *collision between particles* has also not been taken into account. This is an important point of consideration, as in reality, the interaction between particles of the dispersed phase, as well as between the dispersed phase and the continuous phase, cannot be neglected on ground of momentum transfer and consequent energy dissipation. Another physical phenomenon that is closely linked to this process is the *coalescence of particles* (either of the continuous or the dispersed phase), which has not been modelled in the current code. In the case of flows that involve particles that have low surface energy, such as bubbly flows, this is especially important, as coalescence can cause a change in the flow properties, such as local density, pressure etc.
3) The boundary layer that is developed in the current code is a result of the solution obtained from the Navier-Stokes equations, as well as the implementation of the appropriate turbulence model to model the Reynolds stress tensor. A more physically correct solution can be obtained especially near the wall, when the *various layers within the inner boundary layer* (such as the viscous sub-layer and the log-layer) are also included in future editions of the code. This is so that the correct variation of the eddy viscosity is taken into account, along with the corresponding velocity variation. The outer part of the boundary layer can also be modelled in a better manner by incorporating empirical functions to model the variation in wall-normal velocity with distance from the wall.
4) A *more refined grid* can be used for the purpose of studying grid independence tests for various parameters, such as for the velocity profile, pressure etc. A more refined grid will also ensure that the periodic numerical errors that are observed in the current simulation case are reduced, as the interpolation errors would correspondingly diminish as the grid is refined.
5) The current code takes into account isotropic turbulence, which assumes that all the fluctuating components of velocity are equal in magnitude to each other. However, in a real-life case, anisotropy exists in flows. Therefore, a certain degree of *anisotropy* can be introduced by taking into account different models to relate the various fluctuating velocity components.

Appendix

The Python notebooks used for the simulations are given below:

- Simulation of laminar fluid flow - `https://colab.research.google.com/drive/1FugvbGuy4Knc3Mhb0KM6RR-NHrVR38Nh?usp=sharing`
- Simulation of turbulent fluid flow - `https://colab.research.google.com/drive/1aXyk1vaq_N0xo9Ry2sYJeRboK3YISPVZ?usp=sharing`
- Simulation of one-way coupled particulate motion - `https://colab.research.google.com/drive/1q3cgZt3O1QNPt2ZCB0AonhsKzk6bHAdg?usp=sharing`

The fluid flow and particular data collected from the simulations can be found in the link: `https://github.com/shyam97/cmf/tree/Checkpoint-Delta/Data%20Dumps/Total%20Data/Timestamps`

References

[1] Doroshenko, Y., Doroshenko, J., Zapukhliak, V., Poberezhny, L., and Maruschak, P., "Modeling computational fluid dynamics of multiphase flows in elbow and T-junction of the main gas pipeline," *Transport*, Vol. 34, No. 1, 2019, pp. 19–29.

[2] Shams, M., Ahmadi, G., and Smith, D. H., "Computational modeling of flow and sediment transport and deposition in meandering rivers," *Advances in water resources*, Vol. 25, No. 6, 2002, pp. 689–699.

[3] Lahey Jr, R. T., "The simulation of multidimensional multiphase flows," *Nuclear Engineering and Design*, Vol. 235, No. 10-12, 2005, pp. 1043–1060.

[4] Inthavong, K., Tu, J., and Ahmadi, G., "Computational modelling of gas-particle flows with different particle morphology in the human nasal cavity," *The Journal of Computational Multiphase Flows*, Vol. 1, No. 1, 2009, pp. 57–82.

[5] Unverdi, S. O., and Tryggvason, G., "A front-tracking method for viscous, incompressible, multi-fluid flows," 1992.

[6] Unverdi, S. O., and Tryggvason, G., "Computations of multi-fluid flows," *Physica D: Nonlinear Phenomena*, Vol. 60, No. 1-4, 1992, pp. 70–83.

[7] Wörner, M., "Numerical modeling of multiphase flows in microfluidics and micro process engineering: a review of methods and applications," *Microfluidics and nanofluidics*, Vol. 12, No. 6, 2012, pp. 841–886.

[8] Prosperetti, A., and Tryggvason, G., *Computational methods for multiphase flow*, Cambridge university press, 2009.

[9] Brennen, C. E., and Brennen, C. E., *Fundamentals of multiphase flow*, Cambridge university press, 2005.

[10] Crowe, C. T., *Multiphase flow handbook*, Vol. 59, CRC press, 2005.

[11] Mooney, C. Z., *Monte carlo simulation*, Vol. 116, Sage publications, 1997.

[12] Mahadevan, S., "Monte carlo simulation," *Mechanical Engineering-New York and Basel-Marcel Dekker-*, 1997, pp. 123–146.

[13] Kramer, P. R., et al., "A review of some Monte Carlo simulation methods for turbulent systems," *Monte Carlo Methods and Applications*, Vol. 7, No. 3/4, 2001, pp. 229–244.

[14] Bradshaw, P., "Possible origin of Prandt's mixing-length theory," *Nature*, Vol. 249, No. 5453, 1974, pp. 135–136.

[15] Schiller, L., "A drag coefficient correlation," *Zeit. Ver. Deutsch. Ing.*, Vol. 77, 1933, pp. 318–320.

[16] Sommerfeld, M., "Some open questions and inconsistencies of Lagrangian particle dispersion models," *Proc. of 9th Symp. on Turbulent Shear Flows*, Vol. 15, 1993.