Center for Computational Engineering Science Mathematics Division

RWTH AACHEN UNIVERSITY

Diese Arbeit wurde vorgelegt am
Lehrstuhl für Mathematik (MathCCES)
The present work was submitted to
the Chair for Mathematics (MathCCES)

Stochastische Lösungsalgorithmen für Burgers- und Navier-Stokes-Gleichungen

Stochastic solution algorithms for Burgers and Navier-Stokes equations

Masterarbeit
Master thesis
Simulation Sciences

Aachen, November 2019

Vorgelegt von Presented by	Edilbert Christhuraj Koenigstr 21, 52064 Aachen Matrikelnummer: 374762 edilbert.christhuraj@rwth-aachen.de
Erstprüfer First examiner	Prof. Dr. Manuel Torrilhon Lehrstuhl für Mathematik (MathCCES) RWTH Aachen University
Externer Betreuer External Supervisor	Dr. Hossein Gorji Computational Mathematics and Simulation Science (MCSS) École Polytechnique Fédérale de Lausanne

EIGENSTÄNDIGKEITSERKLÄRUNG

Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen und Abbildungen. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer anderen Prüfung vorgelegen.

AFFIDAVIT / STATUTORY DECLARATION

I hereby declare that I wrote this thesis on my own and without the use of any other than the cited sources and tools and all explanations that I copied directly or in their sense are marked as such. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, November 2019

EDILBERT CHRISTHURAJ

CONTENTS

List of Figures v

List of Snippets vi

1 Introduction 1

2 Theory - Stochastic formulation 3
2.1 A stochastic formulation of Burgers equation 3
2.2 A stochastic formulation of Navier-Stokes equation 4

3 Solution algorithms 6
3.1 Particle-grid approach for Burgers equation 6
3.2 Particle-kernel approach for Burgers equation 8
3.3 Particle-grid approach for Navier-Stokes equation 10

4 Implementations of solution algorithms 17
4.1 Particle-grid algorithm for Burgers equation 17
4.2 Implementation of particle-grid algorithm for Burgers equation 17
4.3 Particle-kernel algorithm for Burgers equation 20
4.4 Implementation of particle-kernel algorithm for Burgers equation 21
4.5 Particle-grid algorithm for Navier-Stokes equation 23
4.6 Implementation of particle-grid algorithm for Navier-Stokes equation 23

5 Simulation results 30
5.1 Inviscid Burgers equation 30
5.1.1 Particle-grid: Linear ic - Inviscid - Sample solution . . 30
5.1.2 Particle-grid: Nonlinear ic - Inviscid - Sample solution 31
5.1.3 Particle-grid: Nonlinear ic - Inviscid - Grid convergence 32
5.1.4 Particle-grid: Nonlinear ic - Inviscid - Particle convergence 33
5.1.5 Particle-Kernel: Linear ic - Inviscid - Sample solution 34
5.1.6 Particle-kernel: Nonlinear ic - Inviscid - Sample solution 36
5.1.7 Challenges in the particle-kernel approach for Burgers equation 36
5.2 Viscous Burgers equations 37
5.2.1 Particle-grid: Nonlinear ic - Viscous - Sample solution 37
5.2.2 Particle-grid: Nonlinear ic - Viscous - Grid convergence 38
5.2.3 Particle-grid: Nonlinear ic - Viscous - Particle convergence 39

5.3 Incompressible Navier-Stokes equation 40
5.3.1 Particle-grid: Navier-Stokes - Sample solution 41
5.3.2 Particle-grid: Navier-Stokes - Grid convergence 42

6 Conclusion **44**

Bibliography **45**

LIST OF FIGURES

1 Particle-grid: Domain 6
2 Particle-grid: Discretization 6
3 Particle-grid: Initial cell velocity calculation 7
4 Particle-grid: Particles initialization 7
5 Particle-grid: Cell velocity update 8
6 Particle-grid: Cell velocity update 8
7 Particle-grid: Particles velocity update 8
8 Particle-kernel: Domain 9
9 Particle-kernel: Particles initialization 9
10 Particle-kernel: Particles initial strength calculation 10
11 Particle-kernel: Evolution of particles 10
12 Particle-kernel: Particles velocity update 11
13 Particle-grid: Navier-Stokes - Domain 12
14 Particle-grid: Navier-Stokes - Discretization 12
15 Particle-grid: Navier-Stokes - Initialization of cell velocity and vorticity 13
16 Particle-grid: Navier-Stokes - Particles initialization 14
17 Particle-grid: Navier-Stokes - Evolution of particles 14
18 Particle-grid: Navier-Stokes - Cell velocity update 15
19 Particle-grid: Navier-Stokes - Cell vorticity update 16
20 Particle-grid: Navier-Stokes - Particle velocity and vorticity update 16
21 Particle-grid: Linear ic - Inviscid - Sample solution 31
22 Particle-grid: Nonlinear ic - Inviscid - Sample solution 32
23 Particle-grid: Nonlinear ic - Inviscid - Grid convergence . . . 33
24 Particle-grid: Nonlinear ic - inviscid - particles convergence . 34
25 Particle-kernel: Linear ic - Inviscid - Sample solution 35
26 Particle-kernel: Nonlinear ic - Inviscid - Sample solution . . . 37
27 Particle-grid: Nonlinear ic - Viscous - Sample solution 38
28 Particle-grid: Nonlinear ic - Viscous - Grid convergence . . . 39
29 Particle-grid: Nonlinear ic - Viscous - Particle convergence . . 40
30 Particle-grid: Navier-Stokes - Initial velocity profile 41
31 Particle-grid: Navier-Stokes - Velocity profile at time-step 5 . 42
32 Particle-grid: Navier-Stokes - Grid convergence 43

LIST OF SNIPPETS

1 Particle-grid: Domain setup 17
2 Particle-grid: Particles' velocity initialization 18
3 Particle-grid: Time-step size selection 18
4 Particle-grid: particles' properties initialization 18
5 Particle-grid: Time loop 19
6 Particle-kernel: Domain setup 21
7 Particle-kernel: Particles properties initialization 21
8 Particle-kernel: Particles strength calculation 21
9 Particle-kernel: Time-step size calculation 22
10 Particle-kernel: Time loop 22
11 Particle-kernel: External - Kernel function 23
12 Particle-grid: Navier-Stokes - Domain setup 24
13 Particle-grid: Navier-Stokes - Initial cell velocity and voritcity 24
14 Particle-grid: Navier-Stokes - Particles initialization 25
15 Particle-grid: Navier-Stokes - Time loop 25
16 Particle-grid: Navier-Stokes - External routines 26

ABSTRACT

A wide range of real world problems is formulated in the form of partial differential equations, for example, heat transfer through a solid medium or flow through porous media. Often, these problems are stochastic in nature due to the presence of uncertainty or randomness in the system. In some cases, the mathematical model of the system is not in closed form. To solve such problem, one often uses discretization-based numerical methods such as finite difference or finite volume method. Over the years, many studies have proposed alternative methods. One of such alternatives is the stochastic formulation of a problem and stochastic solution methods. The presence of uncertainty in the problem provides a strong need to consider an approach that incorporates randomness into the numerical method. In this thesis, the stochastic solution algorithms for Burgers and incompressible Navier-Stokes equations are developed. Two approaches, particle-grid and particle-kernel approach, are used for developing the stochastic particle algorithms. These algorithms are demonstrated on one-dimensional Burgers and two-dimensional incompressible Navier-Stokes equations. Convergence studies are performed for each case.

ACKNOWLEDGEMENTS

I sincerely thank Professor Manuel Torrilhon (at MathCCES) for agreeing to be my internal supervisor at my home university, RWTH Aachen. I am grateful to him for his support during the preparation of my application for external master thesis at EPFL.

I would like to thank Professor Jan Hesthaven for approving my request to carry out my master thesis in his research group, Computational Mathematics and Simulation Science (MCSS), at EPFL. From the beginning to the end he has responded to all my questions, clarifications, and concerns swiftly and courteously. I am grateful to him.

I heartily thank my thesis supervisor Dr. Hossein Gorji for providing me this invaluable opportunity to work with him and for making all these things happen in the first place. Starting from our first discussion at AICES kitchen room (on 7^{th} February 2019) till the last day of my report submission (on 5^{th} November 2019) he has been unfailingly supportive of me, in relation to every aspect of my thesis, at all time during that period. I would like to thank him for letting me freely to explore my ideas. And whenever I ran out of ideas, he has provided me new ideas, insightful, often elementary, explanations, and thought-provoking discussions. Finally, and importantly, I am eternally indebted to him for providing me insights beyond the master thesis and for his invaluable support during my search for a doctoral position.

I would like to thank my superb colleagues Mohsen Sadr and Sundar Ganesh for those joyful and memorable times.

Last but not the least, I send my love to my mother and father and to my wonderful sister and thank them for putting up with my excuses.

1 INTRODUCTION

A wide range of real world problems is formulated in the form of partial differential equations, for example, heat transfer through a solid medium or flow through porous media. Often, these problems are stochastic in nature due to the presence of uncertainty or randomness in the system. In some cases, the mathematical model of the system is not in closed form. To solve such problem, one often uses discretization-based numerical methods such as finite difference or finite volume method. Over the years, many studies have proposed alternative methods. One of such alternatives is the stochastic formulation of a problem and stochastic solution methods. The presence of uncertainty in the problem provides a strong need to consider an approach that incorporates randomness into the numerical method.

Representations of solutions of partial differential equations as expected value of functionals of stochastic processes, also known as formalisms, date back to Einstein and Feymann in physics and Kolmogorov and Kac in mathematics [7]. In his work [1], Chorin has made connections between stochastic evolution and deterministic two-dimensional Navier-Stokes equation. The facts: in two-dimensions, the non-linear evolution equation for vorticity obtained from Navier-Stokes has the form of forward Kolmogorov equation and the linear relation between vorticity and velocity given by Biot-Savart law, are used in Chorin's work to represent the vorticity of the Navier-Stokes equation using random walks and a particle limit [7]. This is known as random vortex method [1]. A different approach, a probabilistic approach to two-dimensional incompressible Navier-Stokes equation is given by Busnello in [2]. For three-dimensional incompressible Navier-Stokes equation, a stochastic representation is given in [7].

In this thesis, we consider the stochastic particle approach to calculate the solutions of partial differential equations. Historically, the term stochastic is associated with randomness. Hence, we will see, our methods incorporate the randomness in the numerical method.

Benefits Stochastic particle method has unique characteristics. The benefits of this method are as follows:

- First, given the inherent nature of accommodating randomness in the numerical method, it possible to perform uncertainty quantification analysis with this approach. When modelling a real world problem, we often do not have complete information about the system. Many of the parameters of the model are precarious in nature. Those are associated with uncertainties. One can use this approach to deal with such problems [5].

- Second, extension of stochastic particle approach to higher dimensions

is possible in a straight-forward manner. The algorithmic complexity scales linearly with dimension. In the case of conventional deterministic approaches such as finite difference, finite element method, it is not the case. In this scenario, when extending to higher dimensions, the dimension becomes the power of algorithm complexity.

- Third, the advent of supercomputers and the rapid developments in modern computing architecture have resulted in birth of a new programming paradigm, parallel programming. Stochastic particle approach can harness these technological advancements efficiently, as there are scopes for massive parallelization in this approach.

In addition to the above mentioned benefits there is an additional benefit that is particular to this work. Ultimately, we want to develop a multiscale solver that is completely based on particle methods. For small scales such as nanoscale or microscale there exist particle methods. For example, particles methods are used in molecular dynamics simulations. Particle methods exist for solving Boltzmann equation. On a continuum level, there exists a deterministic particle method, Smoothed Particle Hydrodynamics, to calculate solutions of partial differential equations [10]. But there exist only few literature on stochastic particle methods for solving a continuum partial differential equation.

In this thesis, we consider two partial differential equations, Burgers equation and incompressible Navier-Stokes equation. We solve these equations using stochastic particle approach. To our knowledge, this approach has not been investigated systematically. We have found a relevant work that demonstrates stochastic particle method for solving Burgers equation [6]. However, this work is substantially different from the approach considered in this thesis.

Thesis objective *The main goal of the thesis is to develop procedures to calculate the solutions of one-dimensional Burgers equation and two-dimensional incompressible Navier-Stokes equation using stochastic particle approach.*

The thesis is organized as follows: In the next section, we review the theory of stochastic formulations of Burgers and Navier-Stokes equations. Then we discuss the stochastic solution algorithms in detail. And then a section is devoted to describe the implementation details of the developed algorithms. Simulation results are presented and conclusions are drawn in the subsequent sections.

2 THEORY - STOCHASTIC FORMULATION

We recall few terminologies that are used in this section.

Stochastic process ([3]) A stochastic process is a mathematical object defined as a collection of random variables. These random variables are associated with or indexed by a set of numbers, usually viewed as points in time, giving the interpretation of a stochastic process representing numerical values of some system randomly changing over time, such as growth of bacterial population or the movement of a gas molecule.

Wiener process ([4]) A standard one-dimensional Wiener process, also called Brownian motion, is a stochastic process $\{W_t\}_{t\geq 0+}$ indexed by non-negative real numbers t with the following properties:

- $W_t = 0$.
- With probability 1, the function $t \to W_t$ is continuous in t.
- The process $\{W_t\} \geq 0$ has stationary, independent increments.
- The increment $W_{t+s} - W_s$ has the NORMAL$(0, t)$ distribution.

2.1 A STOCHASTIC FORMULATION OF BURGERS EQUATION

In this section, we present the stochastic formulation of Burgers equation that was originally proposed by Constantin and Iyer [7]. We consider the following inviscid Burgers equation

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = 0 \tag{1}$$

with the initial data

$$u(x, 0) = u_0(x). \tag{2}$$

The above equation describes the evolution of the quantity u. Due to the absence of the pressure term, the quantity u is simply transported by the fluid. So, if we consider X_t as fluid flow map, then $u(X_t)$ is constant in time and $u = u_0(X_t)^{-1}$ gives a formula (method of characteristics) to recover velocity from instantaneous flow trajectory. In Lagrangian formulation, the following equations,

$$\dot{X}_t = u \tag{3}$$
$$A_t = X_t^{-1} \tag{4}$$
$$u = u_0(A_t) \tag{5}$$

with the initial data

$$X(a,0) = a \tag{6}$$

are equivalent to inviscid Burgers equation before the formulation of any shocks. Here, the term X_t^{-1} means spatial inversion of the fluid flow map at time t. We can show that, if we add uniform noise to the fluid flow trajectories and average the resulting noisy flow paths, then we obtain the solution of viscous Burgers equation [7].

Theorem 2.1 ([7]) *Let W be an n-dimensional Wiener process. $k \geq 1$ and $u_0 \in C^{k+1,\alpha}$. Let the pair u, X be a solution of the stochastic system*

$$dX = udt + \sqrt{2\nu}dW \tag{7}$$

$$A = X^{-1} \tag{8}$$

$$u = \boldsymbol{E}[u_0 \circ A] \tag{9}$$

with initial data

$$X(a,0) = a. \tag{10}$$

For boundary conditions, we demand that either u and $X - I$ be spatially periodic or u and $X - I$ decay at infinity. Then u satisfies the viscous Burgers equations

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu\Delta u = 0 \tag{11}$$

with initial data u_0. Here E denotes the expected value with respect to the Wiener measure and I is the identity function.

2.2 A STOCHASTIC FORMULATION OF NAVIER-STOKES EQUATION

We briefly review the stochastic formulation of Navier-Stokes equation that was proposed by Constantin and Iyer [7]. We consider the incompressible Euler equation. It contains the gradient of pressure term. This term maintains the incompressibility in the flow. Hence, in the Lagrangian formulation, while updating the velocity, Leray-Hodge projection method is used to calculate the velocity field transformation under incompressibility condition.

Proposition 2.1 ([7]) *Let $k \geq 1$ and $u_0 \in C^{k+1,\alpha}$ be divergence free. Then u satisfies the incompressible Euler equations*

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u + \nabla p = 0 \tag{12}$$

$$\nabla \cdot u = 0 \tag{13}$$

with initial data u_0 if and only if the pair of functions u and X satisfies the system

$$\dot{X} = u \tag{14}$$

$$A = X^{-1} \tag{15}$$

$$u = \boldsymbol{P}[(\nabla^T A)(u_0 \circ A)] \tag{16}$$

with initial data

$$X(a, 0) = a. \tag{17}$$

Here P is the Leray-Hodge projection on divergence-free vector fields. We impose either periodic boundary conditions and demand that u and $X - I$ be spatially periodic, or demand that u and $X - I$ decay sufficiently rapidly at infinity (I is the identity map).

Theorem 2.2 ([7]) *Let $\nu > 0$, W be an n-dimensional Wiener process, $k \geq 1$, and $u_0 \in C^{k+1,\alpha}$ a given deterministic divergence-free vector field. Let the pair u and X satisfy the stochastic system*

$$dX = u dt + \sqrt{2\nu} dW \tag{18}$$

$$A = X^{-1} \tag{19}$$

$$u = \boldsymbol{EP}[(\nabla^T A)(u_0 \circ A)] \tag{20}$$

with initial data

$$X(a, 0) = a. \tag{21}$$

We impose boundary conditions by requiring that u and $X - I$ either be spatially periodic or decay sufficiently at infinity. Then u satisfies the incompressible Navier-Stokes equations

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu \Delta u + \nabla p = 0 \tag{22}$$

$$\nabla \cdot u = 0 \tag{23}$$

with initial data u_0.

Instead of considering true flow trajectories, if we add noise to the trajectories, then take average of the resulting noisy system, we get the solution to the incompressible Navier-Stokes equation. Proofs can be found in [7].

3 SOLUTION ALGORITHMS

In this section, we present and discuss the solution algorithms for Burgers and Navier-Stokes equations in detail. In this work, we consider two different approaches for developing the algorithms. The first is `particle-grid` approach while the other is `particle-kernel` approach. These two approaches essentially differ from the way we calculate the inverse map during velocity update. As the name suggests, in the particle-grid approach we use `grid` to calculate the inverse map, whereas in particle-kernel we use `kernel` to calculate the same.

3.1 PARTICLE-GRID APPROACH FOR BURGERS EQUATION

We introduce the particle-grid approach for one-dimensional Burgers equation. This algorithm consists totally of six steps.

Step 0 We describe the problem domain. As we solve the one-dimensional problem, the computational domain is one-dimensional and the range of the domain is from 0 to 1, including the boundary points. It is illustrated in Figure 1.

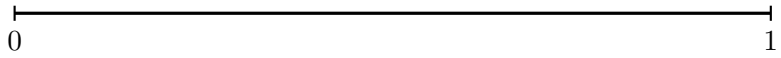

Figure 1: Particle-grid: Domain

Step 1 In the first step, we discretize the domain into equal length of elements called `cells`. We note here that it is not strictly necessary to have equal length for all cells. We choose it for the sake of simplicity. In Figure 2 we illustrate this step by discretizing the domain into 4 cells, each having equal length. Each cell is highlighted by a distinct color.

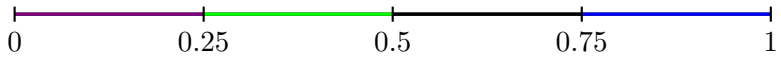

Figure 2: Particle-grid: Discretization

Step 2 Once we discretized the given domain, we proceed to calculate the initial velocity for each cell using the given initial condition. The cell initial velocity is calculated at the centre of each cell. We note again, it is not mandatory to calculate the velocity only at the centre. It can be calculated at any point in the cell. Figure 3 illustrates this step. The centre of each cell is marked by a short vertical line, highlighted in gray color. The cell

velocity is denoted by V. The subscript represent the cell number and the superscript represents the number of time-step. The notation V_2^0 should be read as velocity of the second cell at the zeroth time-step.

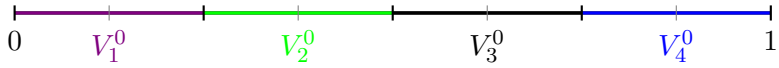

Figure 3: Particle-grid: Initial cell velocity calculation

Step 3 Once we set up our domain and determined the initial properties of the cells, in this step, each cell is populated with particles. These particles are distributed uniformly in the domain. This step is illustrated in Figure 4. Each particle is represented by the purple dot. We note the following details about particles: each particle has a distinct position and a velocity denoted as x and v respectively. The use of superscripts and the subscripts are the same as before. The velocity of a particle is assigned based on its position. Suppose the particle is in the fourth cell, then it gets the velocity of the fourth cell.

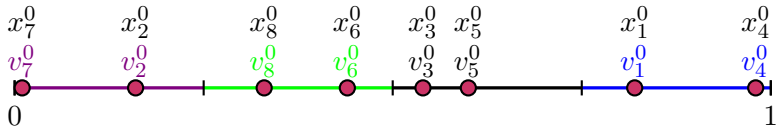

Figure 4: Particle-grid: Particles initialization

Step 4 The system setup is ready. We let the particles to evolve according to the Euler-Maruyama scheme. It is also called the *method of random characteristics* [1].

$$x_i^{n+1} = x_i^n + dtv_i^n + \sqrt{2\nu}\sqrt{dt}\xi_i^n \tag{24}$$

We note the following details about the equation 24: Subscript index i denotes the particle's number: $i = 1, 2, \cdots, N_p$. Total number of particles is denoted by N_p. Superscript index n denotes n^{th} time-step: $n = 1, 2, \cdots, nt$. Total number of time-steps is denoted by nt. The time-step dt is chosen according to the CFL condition. ν is the viscosity. ξ is the standard normal random variable, $\xi \sim \mathcal{N}(0, 1)$. This step is illustrated in Figure 5. We note that the particles positions at the n^{th} time-step. Some particles have moved to other cells.

Step 5 In this step, we update the cell velocity based on the current positions of particles. We refer Figure 6. During the evolution step, some

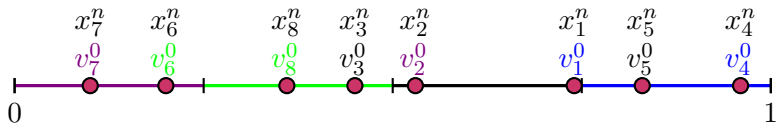

Figure 5: Particle-grid: Cell velocity update

particles moved in from the neighbouring cells. To update the velocity of the cell, we calculate the total velocity for each cell based on the current number of particles in that cell and then divide that by the number of particles that contributed to that calculation.

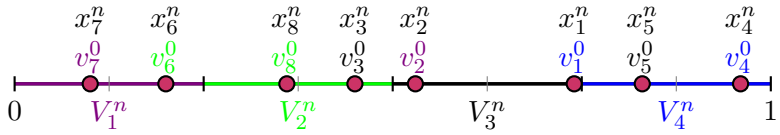

Figure 6: Particle-grid: Cell velocity update

Step 6 In the final step of the algorithm we update the particles velocities. This is done by considering the same procedure we adopted previously. Namely, each particle gets the velocity of the cell they are in. It is illustrated in Figure 7. We note that particles velocities are same as the velocities of the cells they are in. This is highlighted by appropriate colors.

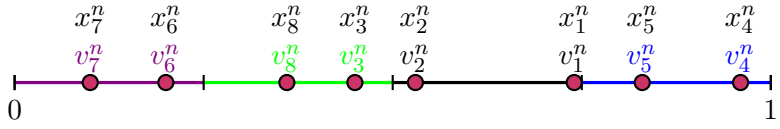

Figure 7: Particle-grid: Particles velocity update

3.2 PARTICLE-KERNEL APPROACH FOR BURGERS EQUATION

In this work, we used a meshfree method for developing the solution algorithm. Instead of using grids, a kernel was employed for the inverse map calculation in the velocity update [8]. In this section, we present the particle-kernel approach for solving one-dimensional Burgers equation.

Step 0 We solve the one-dimensional problem. Hence, we consider the same one-dimensional domain that is described in the previous section. This is shown in Figure 8.

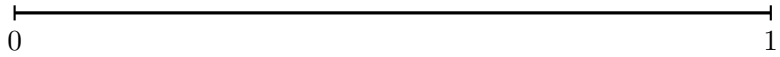

Figure 8: Particle-kernel: Domain

Step 1 As we do not use any grid in this approach, we do not discretize the domain. We populate the entire domain with particles as before. The particles are distributed uniformly in the domain. This step is illustrated in Figure 9. Each particle is represented by the purple dot. We note the following details about the particles: each particle has a distinct position and a velocity denoted as x and v respectively. The velocity of the particles are assigned based on its positions. A smooth velocity function v can be represented as follows [10]:

$$v(x_i^n) = \frac{1}{N_p} \sum_{l=1}^{N_p} w_i^n \delta(x_i^n - x_l^n) \tag{25}$$

We note the following details about the Equation 25: Subscript index i denotes the particle's number: $i = 1, 2, \cdots, N_p$. Total number of particles is denoted by N_p. Superscript index n denotes the nth time-step: $n = 1, 2, \cdots, nt$. Total number of time-steps is denoted by nt. w denotes the weight or strength of the particle. It is discussed in Step 2. δ represents the Gaussian kernel. The kernel is illustrated by the blue curve in the Figure 9.

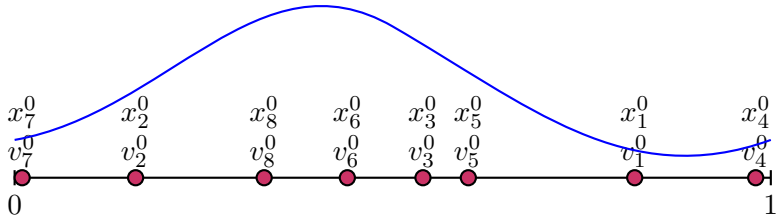

Figure 9: Particle-kernel: Particles initialization

Step 2 In addition to having a position and a velocity, in this approach, each particle has one more property called *weight* or *strength*. The initial strength of the particles is determined based on the initial condition under consideration. It is shown in Figure 10. The strengths of the particles are denoted by w. To determine the initial strengths, we use the velocity equation mentioned in the previous step. At time $t = 0$ the velocity equation reads

$$v(x_i^0) = \frac{1}{N_p} \sum_{l=1}^{N_p} w_i^0 \delta(x_i^0 - x_l^0) \tag{26}$$

We solve this equation for w^0. This means, we need to solve the following linear system of equations:

$$w^0 = A^{-1}v^0 \tag{27}$$

Once we calculated the w at the initial time $t = 0$, we use the same strengths of the particles throughout our simulation. The strengths are constants. Hence, there is no necessity to recalculate it at every time step.

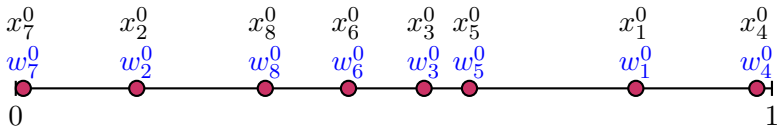

Figure 10: Particle-kernel: Particles initial strength calculation

Step 3 Once we set up the system, we let the particles to evolve according to the Euler-Maruyama scheme as before.

$$x_i^{n+1} = x_i^n + dtv_i^n + \sqrt{2\nu}\sqrt{dt}\xi_i^n \tag{28}$$

We note the following details about the Equation 28: Subscript index i denotes the particle's numbe:r $i = 1, 2, \cdots, N_p$. Total number of particles is denoted by N_p. Superscript index n denotes the n^{th} time-step: $n = 1, 2, \cdots, nt$. Total number of time-steps is denoted by nt. ν is the viscosity. ξ is the standard normal random variable, $\xi \sim \mathcal{N}(0, 1)$. In Figure 11, we note that the particles have moved to different positions.

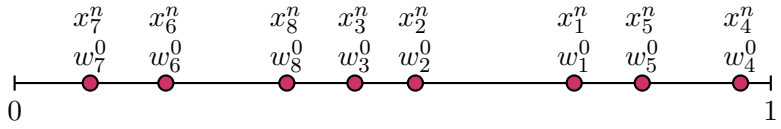

Figure 11: Particle-kernel: Evolution of particles

Step 4 In this final step of the algorithm, we update the velocities of the particles. To calculate the new particle velocity, we use the Equation 25. In the Step 2, we determined the initial strengths of the particles. We use the Gaussian kernel δ around the particle for which the velocity needs to be updated. This step is shown in Figure 12.

3.3 PARTICLE-GRID APPROACH FOR NAVIER-STOKES EQUATION

In this section, we present the particle-grid approach algorithm for solving two-dimensional incompressible Navier-Stokes equation. The algorithm consists totally of seven steps. Before we discuss the algorithm, we introduce the

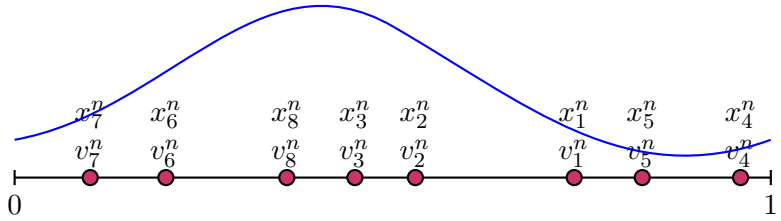

Figure 12: Particle-kernel: Particles velocity update

following details: We solve the following 2D incompressible Navier-Stokes equation

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u + \nabla p - \nu \Delta u = 0 \tag{29}$$

$$\nabla \cdot u = 0 \tag{30}$$

$$u(x, 0) = u_0(x). \tag{31}$$

Taking the curl operator with the above equation ($\nabla \times \bullet$) leads to the following equation

$$\frac{\partial \omega}{\partial t} + (u \cdot \nabla)\omega - \nu \Delta \omega = 0 \tag{32}$$

$$\omega(x, 0) = \omega_0(x). \tag{33}$$

As a result of taking the curl operation on Navier-Stokes equation, the curl of gradient of the pressure term vanishes, and we know the curl of velocity is the vorticity. Hence, we get an evolution equation for the vorticity ω. This evolution equation is similar to the form of viscous Burgers equation that we saw in the previous section, for which we developed the solution algorithm. The difference is the solution of the above equation is not the velocity u but the vorticity w. The idea is to use the previously developed algorithm to solve the vorticity evolution equation and then we use the 2D Biot-Savart law to recover the velocity from vorticity [11]. We define the two-dimensional Biot-Savart equation later in this section.

Step 0 As we want to solve the two-dimensional Navier-Stokes equation, we consider the following two-dimensional unit square grid as shown in Figure 13.

Step 1 In the particle-grid approach, we discretize the two-dimensional domain into N number of cells that are equal in size. We note that we use an equidistant grid. That means, grid spacing in the "x" direction (horizontal) and in the "y" direction (vertical) are the same. In other words, $dx = dy$.

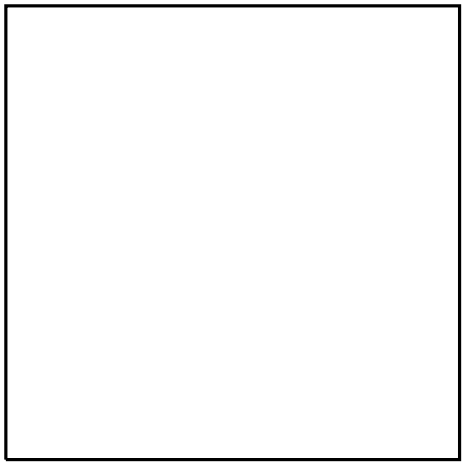

Figure 13: Particle-grid: Navier-Stokes - Domain

But this need not be the case. We have chosen this for the sake of simplicity. One can choose any type of grid spacing as the application dictates. This step is illustrated in Figure 14. For demonstration purpose, we discretize the domain into four cells of equal size. Each cell is highlighted with a distinct color.

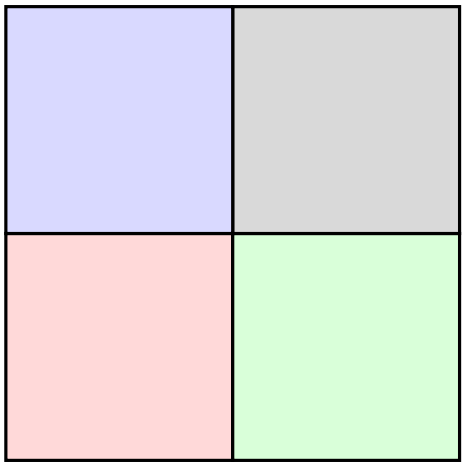

Figure 14: Particle-grid: Navier-Stokes - Discretization

Step 2 Once we discretized the domain, we calculate the properties (velocity and vorticity) of each cell. We calculate the cell velocity at the centre of each cell. For the calculation of vorticity, we can use any available technique. If analytical formula exists, we can directly use the formula to calculate the initial vorticity of the cell. When there exists no analytical expression for vorticity calculation, we then resort to numerical approximation for the cal-

culations. Figure 15 illustrates this step. The centre of each cell is marked with a small black dot. And the letters V and Ω denotes the cell velocity and cell vorticity respectively. The subscripts and the superscripts are the same as mentioned in the previous section.

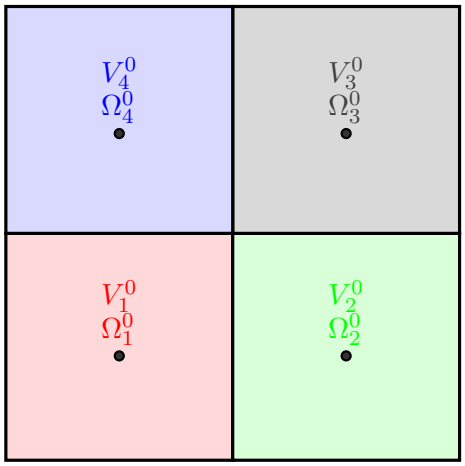

Figure 15: Particle-grid: Navier-Stokes - Initialization of cell velocity and vorticity

Step 3 Once we set up our computational domain, we then initialize each cell with particles. The particles are distributed uniformly across the domain. We note the following details about particle initialization: At the initial time-step each cell has equal number of particles. All particles have three properties (position, velocity and vorticity). Each particle gets the velocity and vorticity of the respective cell they are in. We denote the particles position by x, velocity by v and vorticity by ω. We note here that the x is vector of two dimension, x and y. I It is the same for velocity. It is a vector having two components v_x and v_y. This step is illustrated in Figure 16. For the sake of clarity, we omitted the individual vector components.

Step 4 The system setup is complete. We let the particles in the system to evolve according to Euler-Maruyama scheme. The equations are:

$$x_i^{n+1} = x_i^n + dtv_i^n + \sqrt{2\nu dt}\xi_i^n \tag{34}$$

$$y_i^{n+1} = y_i^n + dtv_i^n + \sqrt{2\nu dt}\xi_i^n. \tag{35}$$

We note the following details about the Equation 34 and the Equation 35: Subscript i represents the particle's number: $i = 1, 2, \cdots, N_p$. Total number of particles is denoted by N_p. Superscript n denotes the n^{th} time-step: $n = 1, 2, \cdots, nt$. Total number of time-steps is denoted by nt. The time-step dt is chosen according to the CFL condition. As this is the two-dimensional

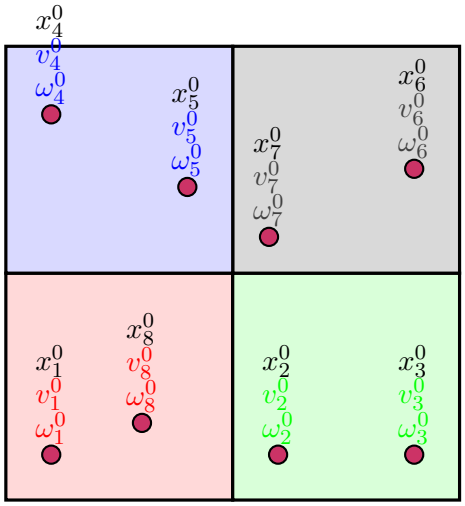

Figure 16: Particle-grid: Navier-Stokes - Particles initialization

case, we select the least of two dimensions. ξ represents the standard two-dimensional normal random variable. Figure 5 represents the illustration of this step. We observe that after this step, particles have moved to new positions. Some particles are in the same cell while some are in the new cells.

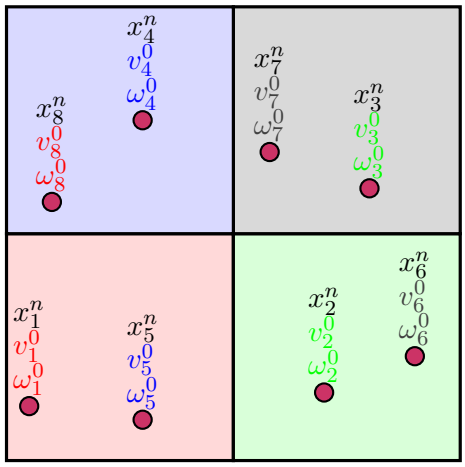

Figure 17: Particle-grid: Navier-Stokes - Evolution of particles

Step 5 After the evolution step we update the cell properties. In this step, we discuss updating the velocity of the cell. As mentioned earlier in the section, we use two-dimensional Biot-Savart law to recover the velocity

from vorticity. The two-dimensional Biot-Savart equation reads as [11]:

$$V(X) = \frac{1}{N_p} \sum_{i=1}^{N_p} \omega_i K(X - x_i). \tag{36}$$

Where the kernel $K(r)$ is defined as

$$K(r) = \frac{1}{2\pi|r|^2} \quad \begin{bmatrix} -r_2 \\ r_1 \end{bmatrix}.$$

In the above equation, N_p represents the total number of particles in the system. This formula gives the velocity value at the centre of the cell X based on the current positions and vorticities of the particles. In Figure 18 the V_1^n reads as the velocity of cell one at the n^{th} time-step. We note that Equation 36 does not take into account the effect of boundary conditions. Therefore, for simplicity we consider the free boundary setting for the rest of this study.

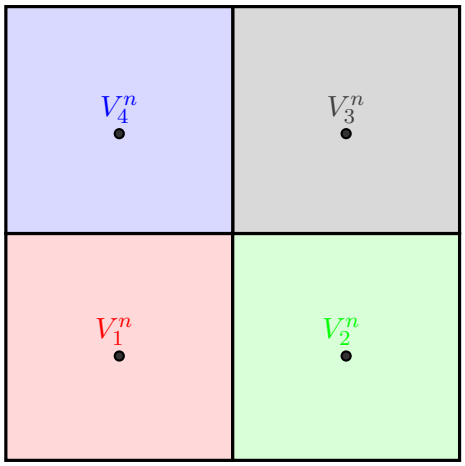

Figure 18: Particle-grid: Navier-Stokes - Cell velocity update

Step 6 Once we updated the cell velocity, we update the cell vorticity Ω. We use the first order approximation for the update. We calculate the total vorticity for each cell by summing the vorticity contributions from the particles in that cell. We then divide the total vorticity by the number of particles that contributed to this sum. Essentially, we take cell averages for the vorticity update. This step is illustrated in Figure 19.

Step 7 In the final step of this algorithm, we consider updating the properties (velocity and vorticity) of the particles. For each particle, we assign the velocity and the vorticity of the respective cell they are in, at current

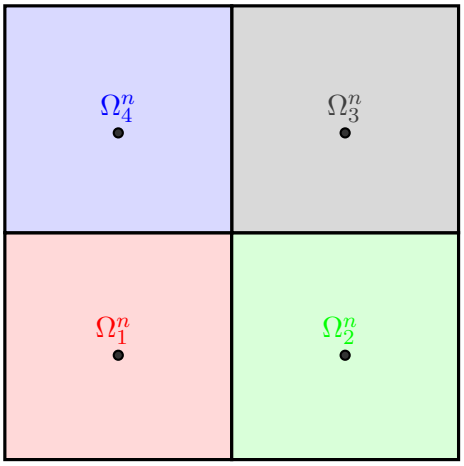

Figure 19: Particle-grid: Navier-Stokes - Cell vorticity update

time-step, as we did previously. This is shown in Figure 20. We note the colors of the particles properties. It matches the color of the cells they are in.

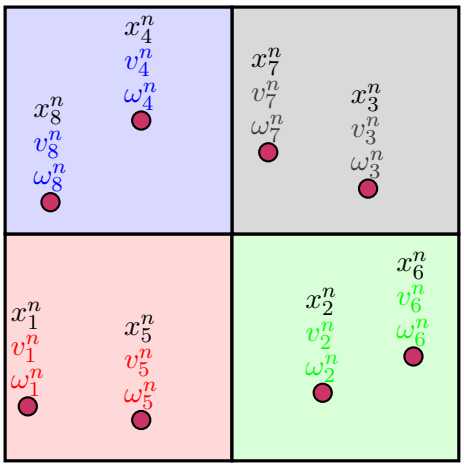

Figure 20: Particle-grid: Navier-Stokes - Particle velocity and vorticity update

4 IMPLEMENTATIONS OF SOLUTION ALGORITHMS

In this section, we provide the implementations of the stochastic algorithms for Burgers and Navier-Stokes equations in MATLAB. There are no particular reasons for selecting this language over others. However, with this intuitive implementation at hand, one can easily rewrite these algorithms in any high performance languages.

4.1 PARTICLE-GRID ALGORITHM FOR BURGERS EQUATION

We summarize the important steps involved in the particle-grid approach for Burgers equation in Algorithm 1. The provided matlab implementation will follow this algorithm. We note here, readability and clarity are preferred to performance. One might find a more efficient way of implementing the same algorithm in a different platform. But this meant to provide a comprehensive understanding of the algorithm.

Algorithm 1: Particle-grid algorithm for Burgers equation

```
  Input: domain, viscosity, grid space, number of
         cells and particles, final time, initial
         condition, boundary condition
  Output: solution at a given time
1 domain set up
2 initial cell velocity
3 particles' initialization
4 while t < final time do
5 |  particles' position update
6 |  if particle position is outside of boundary then
7 |  |  enforce periodic bc     // bc - boundary condition
8 |  cell velocity update
9 |  particles' velocity update
```

4.2 IMPLEMENTATION OF PARTICLE-GRID ALGORITHM FOR BURGERS EQUATION

In the following, we divided the complete source code into short snippets, and we provided comments and remarks for each snippet.

Snippet 1: Particle-grid: Domain setup

```
1 xStart = 0; xEnd = 1; % Range of Domain
2 nb_cells = 1000;  % Number of cells
3 nb_particles = 100; % Number of particles per cell
```

```
x = linspace(xStart, xEnd, nb_cells + 1); % discretize into
    cells
dx = x(2) - x(1); % grid size
%% ****** Particles ******
total_nb_particles = nb_particles * nb_cells;
par_old = zeros(3, total_nb_particles); % particle history -old
```

The one-dimensional domain setup for particle-grid approach is presented in Snippet 1. The domain is from 0 to 1 and the grid space dx is uniform.

Snippet 2: Particle-grid: Particles' velocity initialization

```
%% ****** Cell centre calculation ******
cell_centre = zeros(1, nb_cells);
for i = 1:nb_cells
    cell_centre(i) = x(i) + (dx / 2);
end
%% ****** Memory allocation for cell velocity *******
cell_vel = 0 .* cell_centre;  % old cell velocity
new_cell_vel = 0 .* cell_vel; % New cell velocity
particles = 0 .* cell_centre; % current cell numbers of
    particles
%% ****** Cell velocity initialization ********
cell_vel=sin(2*pi*cell_centre);
```

In Snippet 2 cell velocity initialization is presented. We note that the use of `new cell velocity` variable facilitates us in storing the velocity value at the current time step.

Snippet 3: Particle-grid: Time-step size selection

```
%% time-step
CFL = 0.9;
final_time = 0.1; % final time
dt = CFL * dx / max(cell_vel)
nt = uint32(final_time/dt)
```

Snippet 3 illustrates the calculation for selecting the correct time-step size according to the CFL condition. The user sets the CFL value and the step size is automatically calculated based on the grid size and the maximum velocity.

Snippet 4: Particle-grid: particles' properties initialization

```
%%  ******* Particles position initialization *******
```

```
for i = 1:nb_cells
    for j = (((i - 1) * nb_particles) + 1):(i * nb_particles)
        par_old(1, j) = x(i) + dx * rand(1, 1); % particle
            position
    end
end
%% ****** Particles velocity initialization *******
for i = 1:nb_cells
    for j = (((i - 1) * nb_particles) + 1):(i * nb_particles)
        par_old(2, j) = cell_vel(1, i); % particle velocity
        par_old(3, j) = i; % particle cell number
    end
end
% Copy current particle history to the variable for manipulation
par_new = par_old;
```

Particles' properties initialization is presented in Snippet 4. For the uniform distribution of particles we use the MATLAB inbuilt function *rand* which produces a uniform random number. Based the position of particles the code assigns the velocity to each particle. In the initial time-step, as the position of the particle is known, we do not need a search algorithm to find it.

Snippet 5: Particle-grid: Time loop

```
for time = 1:nt
   %Position update
    for i = 1:total_nb_particles
        par_new(1, i) = par_old(1, i) + (dt * par_old(2,i)) + (
            sqrt(2*dt*nu)*randn(1,1));
        if (par_new(1, i) > xEnd)
            par_new(1, i) = par_new(1, i) - xEnd;
        elseif (par_new(1, i) < xStart)
            par_new(1, i) = par_new(1, i) + xEnd;
        end
    end
    %Cell velocity update
    for i = 1:total_nb_particles
        position_x = par_new(1, i);
        xl = 1;
        xr = nb_cells + 1;
        xm = floor((xl + xr) / 2);
        [xl, xr] = find_cell_x(position_x, xl, xr, xm, x);
        par_new(3, i) = xl;
```

```
19      end
20      new_cell_vel = zeros(1, nb_cells);
21      particles = zeros(1, nb_cells);
22      for i = 1:total_nb_particles
23          c = par_new(3, i);
24          new_cell_vel(1, c) = new_cell_vel(1, c) + par_new(2, i);
25          particles(1, c) = particles(1, c) + 1;
26      end
27      for i = 1:nb_cells
28          if (particles(1, i) == 0)
29              new_cell_vel(1, i) = 0;
30          else
31              new_cell_vel(1, i) = new_cell_vel(1, i) / particles
                    (1, i);
32          end
33      end
34      % particle velocity update
35      for i = 1:total_nb_particles
36          c = par_new(3, i);
37          par_new(2, i) = new_cell_vel(1, c);
38      end
39      par_old = par_new;
40  end
```

Snippet 5 shows the main loop of the algorithm. This contains the time loop of the algorithm, particles' position update, cell velocity update and the particles' velocity update. The code executes as follows: It updates the particles positions. Then a search-algorithm is used to find the cell number of particles based on its new positions. Then the cell average velocity is calculated and updated. And finally, the particles' velocity are updated accordingly. The main idea is to store the particles' cell number once and it can be used at a later step.

4.3 PARTICLE-KERNEL ALGORITHM FOR BURGERS EQUATION

The particle-kernel algorithm for one-dimensional Burgers equation is summarized in this section. The MATLAB implementation very closely follows these steps.

Algorithm 2: Particle-kernel algorithm for Burgers equation

```
Input: domain, viscosity, number of particles,
       final time, initial condition, boundary
       condition
Output: solution at a given time
domain set up
particles' position initialization
particles' strength calculation
while t < final time do
    particles' position update
    if particle position is outside of boundary then
        enforce periodic bc     // bc - boundary condition
    particles' velocity calculation
```

4.4 IMPLEMENTATION OF PARTICLE-KERNEL ALGORITHM FOR BURGERS EQUATION

Snippet 6: Particle-kernel: Domain setup

```
xStart=0; xEnd=1;
nb_cells=1000; nu=0;
nb_particles=1000;
initial_velocity = zeros(1,nb_particles);
new_vel = 0 .* initial_velocity;
par = zeros(3,nb_particles);
```

Snippet 6 shows the code for setting up the one-dimensional domain. As this is a meshfree methods, we do not discretize the domain.

Snippet 7: Particle-kernel: Particles properties initialization

```
for i=1:nb_particles
    par(1,i)=rand; % particles position
    par(2,i)=sin(2*pi*par(1,i)); % particle velocity
end
ip = par(1,:);
initial_velocity = par(2,:);
```

In Snippet 7 the properties of particles such as positions and and velocities are initialized.

Snippet 8: Particle-kernel: Particles strength calculation

```
matrix = zeros(nb_particles,nb_particles);
for i=1:nb_particles
    for j=1:nb_particles
        matrix(i,j)=phi((par(1,i)-par(1,j))); % Matrix-M
    end
end
matrix = (matrix ./ nb_particles) ;% .* (sqrt(pi)*nb_particles
    *10^(-2));
W=matrix\par(2,:)';
par(3,:)=W'; % particle weight
```

Snippet 8 shows the calculation of the initial strengths of the particles. This is obtained by solving the set of linear system of equations.

Snippet 9: Particle-kernel: Time-step size calculation

```
par_new =par;
W = W';
dt =10^(-4);
tf = 0.14;
nt = uint32(tf/dt);
```

As this method involves no discretization, the time step should be provided. The time-step that is smaller than the inverse value of total number of particles in the system is prescribed.

Snippet 10: Particle-kernel: Time loop

```
for time=1:nt
    for i=1:nb_particles
        par_new(1,i) = par(1,i) + dt* par(2,i)+ sqrt(2*nu*dt)*
            randn(1,1);
    end
    for i=1:nb_particles
        for j=1:nb_particles
            par_new(2,i)=par_new(2,i) + phi((par_new(1,i)-
                par_new(1,j)) )* par_new(3,j);
        end
        par_new(2,i)=par_new(2,i)./nb_particles;
    end
    par = par_new;
    for i=1:nb_particles
        par_new(2,i)=0;
    end
end
```

Snippet 10 shows the main loop of the algorithm which involves the position update and the particles' velocity calculation steps.

Snippet 11: Particle-kernel: External - Kernel function

```
function y = phi(value)
eps =10^(-7);
if (abs(value) < 2*eps )
    y = (1/(eps*sqrt(pi))) * ( exp(-(value/eps)^2) );
else
    y =0;
end
end
```

Finally, Snippet 11 presents the kernel routine that is used in the calculation of the particles' velocity in the main loop. It is the Gaussian kernel. The cut off function *epsilon* is used to control the width of the kernel.

4.5 PARTICLE-GRID ALGORITHM FOR NAVIER-STOKES EQUATION

In this section, the particle-grid algorithm for two-dimensional incompressible Navier-Stokes equation is summarized in Algorithm 3.

Algorithm 3: Particle-grid algorithm for Navier-Stokes equation

```
Input: domain, viscosity, grid space, number of
       cells and particles, final time, initial
       condition, boundary condition
Output: solution at a given time
domain set up
initial cell velocity
initial cell vorticity
particles' initialization
while t < final time do
    particles' position update
    if particle position is outside of boundary then
        enforce periodic bc     // bc - boundary condition
    cell velocity update
    cell vorticity update
    particles' velocity and vorticity update
```

4.6 IMPLEMENTATION OF PARTICLE-GRID ALGORITHM FOR NAVIER-STOKES EQUATION

Snippet 12: Particle-grid: Navier-Stokes - Domain setup

```
%**********2D domain and particles' position initialization ***
xStart = 0; xEnd = 1;
yStart = 0; yEnd= 1;
nb_cells_in_x = 20;
nb_cells_in_y = 20;
[total_nb_cells,x_domain,y_domain,dx,dy]=DISCRETIZE (xStart,
    xEnd,yStart,yEnd,nb_cells_in_x,nb_cells_in_y);
[x_cell,y_cell] = CELL_LENGTH(nb_cells_in_x,nb_cells_in_y,
    x_domain,y_domain);
nb_of_particles_in_a_cell = 100;
total_nb_particles= nb_of_particles_in_a_cell*nb_cells_in_x*
    nb_cells_in_y;
par_new = zeros(5,total_nb_particles);
diff_co_eff=0.01;
cell_vel = zeros(2,total_nb_cells);
new_cell_vel = zeros(2,total_nb_cells);
new_cell_curl = zeros(1,total_nb_cells);
current_cell = zeros(1,total_nb_particles);
num_par_in_a_cell=0;
pp = zeros(1,total_nb_cells);
[cell_coord, cell_centre_coord] = CELL_COORDINATES(x_domain,
    y_domain,nb_cells_in_x,nb_cells_in_y,total_nb_cells);
par_old = INIT_POS_PAR(total_nb_cells, nb_of_particles_in_a_cell
    ,cell_centre_coord,total_nb_particles,dx,dy);
```

In Snippet 12, the two-dimensional domain setup is presented. We note the following: The grid spacing in each dimension, x and y, are separately provided. Hence, non-uniform grid can be chosen as desired. The details of the coordinates of each corner of the cell and the centre of the cell are stored for later use.

Snippet 13: Particle-grid: Navier-Stokes - Initial cell velocity and voritcity

```
%******Cell properties ************
for i=1:total_nb_cells
        cell_vel(1,i)=sin(2*pi* cell_centre_coord(2,i));
        cell_vel(2,i)=cos(2 *pi*cell_centre_coord(1,i));
        cell_curl(1,i)= -( 2*pi*cos(2 *pi*cell_centre_coord(2,i)
            ) )-( 2*pi*sin(2*pi*cell_centre_coord(1,i)) ) ;
end
%**********TIME STEP*************
CFL=0.9;
final_time=0.0005;
```

```
Xmax=max(cell_vel(1,:));
Ymax=max(cell_vel(2,:));
if Xmax>Ymax
    dt = (CFL*dx) / Xmax;
elseif Ymax>Xmax
    dt = (CFL*dx) / Ymax;
else
    dt = (CFL*dx) /Xmax;
end
```

Snippet 13 shows the code for the initialization of cell properties such as velocity and vorticity. It also shows the time-step size selection. The selection is done based on the given CFL initial condition and the grid spacing in each dimension and the maximum velocity.

Snippet 14: Particle-grid: Navier-Stokes - Particles initialization

```
% Initialize each particle's velocity and vorticity
for i = 1:total_nb_cells
    for xInd=(((i-1)*nb_of_particles_in_a_cell)+1):(i*
        nb_of_particles_in_a_cell)
        par_old(3,xInd)=cell_vel(1,i);
        par_old(4,xInd)=cell_vel(2,i);
        par_old(5,xInd)=cell_curl(1,i);
    end
end
% copy for manipulation
par_new(1,:) = par_old(1,:);
par_new(2,:) = par_old(2,:);
par_new(5,:) = par_old(5,:);
t=0;
```

The initialization of particles' properties such as velocity and vorticity are presented in Snippet 14. As the initial positions are known, a search-algorithm is not needed to assign the properties.

Snippet 15: Particle-grid: Navier-Stokes - Time loop

```
for time=0:dt:final_time
    t=t+1;
    % update the particle position
    [par_new] = POS_UPDATE(total_nb_particles,par_old,par_new,
        diff_co_eff,dt,...
        xStart,xEnd,yStart,yEnd);
    % update cell vorticity
```

```
7       [pp,current_cell,new_cell_curl]= NEW_CELL_CURL (current_cell
            ,total_nb_particles,total_nb_cells,nb_cells_in_x, ...
8           nb_cells_in_y,x_domain,y_domain,par_new,new_cell_curl,pp
                );
9       % update cell velocity
10      new_cell_vel = 0 .* new_cell_vel;
11      for j=1:total_nb_cells
12          for i=1:total_nb_particles
13              dist_x = cell_centre_coord(1,j)-par_new(1,i);
14              dist_y = cell_centre_coord(2,j)-par_new(2,i);
15              modulo = dist_x^2+dist_y^2;
16              new_cell_vel(1,j)= new_cell_vel(1,j) + par_new(5,i)*
                    Kx(modulo,dist_y);
17              new_cell_vel(2,j)= new_cell_vel(2,j) + par_new(5,i)*
                    Ky(modulo,dist_x);
18          end
19          new_cell_vel(1,j) = (1/total_nb_particles) *
                new_cell_vel(1,j);
20          new_cell_vel(2,j) = (1/total_nb_particles) *
                new_cell_vel(2,j);
21      end
22      % particles' properties  Update
23      for i=1:total_nb_particles
24          r = current_cell(1,i);
25          par_old(3,i)= new_cell_vel(1,r);
26          par_old(4,i)= new_cell_vel(2,r);
27          par_new(5,i)= new_cell_curl(1,rh
28      end
29  end
```

Snippet 15 shows the main loop of the algorithm. It consists of updating particles' position, cells' velocity and vorticity and particles' velocity and vorticity. After updating the particles position, the search algorithm finds the current cell number of particles based on its new positions. This will later be used in the calculations.

In the following we present the external routines that are used in the main program.

Snippet 16: Particle-grid: Navier-Stokes - External routines

```
1  %% ****Biot-savart kernel**********
2  function vector = Kx(modulo,value)
3  vector = (1/(2*pi*modulo))*(-(value));
4  end
5  function vector = Ky(modulo,value)
```

```
vector = (1/(2*pi*modulo))* ((value));
end
%% **** Discretization *********
function [total_nb_cells,x_domain,y_domain,dx,dy]=DISCRETIZE (
    xStart,xEnd,yStart,yEnd,nb_cells_in_x,nb_cells_in_y)
total_nb_cells = nb_cells_in_x* nb_cells_in_y;
x_domain= linspace(xStart,xEnd,nb_cells_in_x+1);
y_domain= linspace(yStart,yEnd,nb_cells_in_y+1);
dx = x_domain(2)-x_domain(1);
dy = y_domain(2)-y_domain(1);
end
%% ***** Cell lengths *****
function [x_cell,y_cell] = CELL_LENGTH(nb_cells_in_x,
    nb_cells_in_y,x_domain,y_domain)
x_cell = zeros(nb_cells_in_x);
y_cell = zeros(nb_cells_in_y);
for index=1:nb_cells_in_x
    x_cell(index)=x_domain(index+1);
end
for index=1:nb_cells_in_y
    y_cell(index)=y_domain(index+1);
end
end
%% ****Cell coordinates calculations *****
function [cell_coord, cell_centre_coord] = ...
    CELL_COORDINATES(x_domain,y_domain,nb_cells_in_x,
        nb_cells_in_y,total_nb_cells)
    cell_coord = zeros(5,total_nb_cells);
    cell_centre_coord = zeros(5,total_nb_cells);
for i=1:nb_cells_in_x %pointer is at column and progress through
     row
    for j=i:nb_cells_in_x:total_nb_cells
        cell_coord(1,j)=x_domain(1,i);
        cell_coord(2,j)=x_domain(1,i+1);
        cell_centre_coord(1,j)=x_domain(1,i)+ (x_domain(1,i+1)-
            x_domain(1,i))/2;
        cell_coord(5,j)=j;
    end
end
for i=1:nb_cells_in_y %pointer is at row and progress through
    row
    for j= (((i-1)*nb_cells_in_x)+1):(i*nb_cells_in_x)
        cell_coord(3,j)=y_domain(1,i);
        cell_coord(4,j)=y_domain(1,i+1);
```

```
        cell_centre_coord(2,j)=y_domain(1,i)+ (y_domain(1,i+1)-
            y_domain(1,i))/2;
    end
end
end
%% ***** Particles position initialization ******
function par_old = INIT_POS_PAR(total_nb_cells,
    nb_of_particles_in_a_cell, cell_centre_coord,
    total_nb_particles,dx,dy)
par_old = zeros(1,total_nb_particles);
for i=1:total_nb_cells
    for j=(((i-1)*nb_of_particles_in_a_cell)+1):(i*
        nb_of_particles_in_a_cell)
        par_old(1,j)=cell_centre_coord(1,i) + dx * rand(1,1) ; %
            Uniform distribution
    end
end
for i=1:total_nb_cells
    for j=(((i-1)*nb_of_particles_in_a_cell)+1):(i*
        nb_of_particles_in_a_cell)
        par_old(2,j)=cell_centre_coord(2,i) + dy * rand(1,1); %
            uniform distribution
    end
end
end
%% **** Particles position update *********
function [par_new] = POS_UPDATE (total_nb_particles,par_old,
    par_new,diff_co_eff,dt,xStart,xEnd,yStart,yEnd)
for i=1:total_nb_particles
    par_new(1,i)=par_old(1,i)+dt * par_old(3,i) + sqrt(2*
        diff_co_eff)* sqrt(dt)*randn;
    %periodic boundary condition
    if par_new(1,i) > xEnd
        par_new(1,i) = par_new(1,i)-xEnd;
    elseif par_new(1,i) < xStart
        par_new(1,i) = par_new(1,i)+xEnd;
    end
    par_new(2,i)=par_old(2,i)+dt * par_old(4,i) + sqrt(2*
        diff_co_eff)* sqrt(dt)*randn;
    if par_new(2,i) > yEnd
        par_new(2,i) = par_new(2,i)-yEnd;
    elseif par_new(2,i) < yStart
        par_new(2,i) = par_new(1,i)+yEnd;
    end
```

```
78  end
79
80  end
81  %% *** Cell vorticity calculation ***
82  function [pp,current_cell,new_cell_curl]= NEW_CELL_CURL (
        current_cell,total_nb_particles,total_nb_cells,nb_cells_in_x
        , nb_cells_in_y,x_domain,y_domain,par_new,new_cell_curl,pp)
83  num_par_in_cell= zeros(1,total_nb_cells);
84  new_cell_curl = 0.*new_cell_curl;
85  position_x =0; position_y=0;
86  for par_number=1:total_nb_particles
87      xl=1; xr=nb_cells_in_x+1; xm=floor((xl+xr)/2);
88      position_x = par_new(1,par_number);
89      [xl,xr] = find_cell_x(position_x,xl,xr,xm,x_domain);
90      position_y = par_new(2,par_number);
91      yd=1; yu=nb_cells_in_y+1; ym=floor((yd+yu)/2);
92      [yd,yu] = find_cell_y(position_y,yd,yu,ym,y_domain);
93      current_cell(1,par_number)=(xl+(yd-1)*nb_cells_in_x);
94      new_cell_curl( (xl+(yd-1)*nb_cells_in_x) ) =  ...
95          new_cell_curl((xl+(yd-1)*nb_cells_in_x)) + par_new(5,
              par_number);
96      num_par_in_cell((xl+(yd-1)*nb_cells_in_x))= ...
97          num_par_in_cell((xl+(yd-1)*nb_cells_in_x))+1;
98  end
99  new_cell_curl = new_cell_curl ./ num_par_in_cell;
100 pp = num_par_in_cell;
101 end
```

5 SIMULATION RESULTS

In this section, we present the simulation results of the algorithms developed in the previous section and discuss the results and some of the faced challenges in detail.
This section is divided into three subsections: we discuss the results of inviscid Burgers equation and then we proceed to discuss the results of viscous Burgers equation. Finally, we discuss the results of incompressible Navier-Stokes equations. For the sake of brevity, initial condition is abbreviated as *ic* throughout this section.

5.1 INVISCID BURGERS EQUATION

Two approaches, the particle-grid and the particle-kernel approaches, were tested on one-dimensional inviscid Burgers equation. Two test cases were considered: linear initial condition and nonlinear initial condition. For each case, the following details are presented: the model problem, some of the simulation parameters, and the result.

5.1.1 PARTICLE-GRID: LINEAR IC - INVISCID - SAMPLE SOLUTION

Model Problem We consider the following inviscid Burgers equation with linear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = 0$$
$$u(x, 0) = x$$

Simulation data The following parameters are used in the simulation to obtain the results.

- free boundary condition
- $\nu = 0$
- $CFL = 0.5$
- Number of cells $N = 1000$
- Number of particles per cell = 30

Result The result of the above simulation setup is presented in Figure 21. The x-axis represents the position and the y-axis represents the function value. The blue line marks the initial condition. The red line marks the solution obtained by method of characteristics. And the yellow line marks the solution obtained by the stochastic particle-grid approach. We see that the stochastic result matches very accurately with the exact solution.

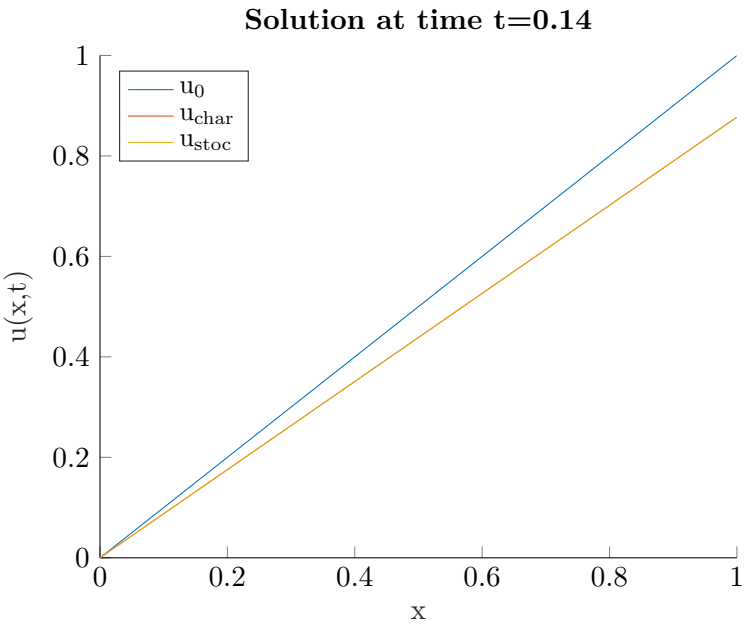

Figure 21: Particle-grid: Linear ic - Inviscid - Sample solution

5.1.2 PARTICLE-GRID: NONLINEAR IC - INVISCID - SAMPLE SOLUTION

Model Problem We consider the following inviscid Burgers equation with nonlinear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = 0$$
$$u(x, 0) = sin(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- periodic boundary condition
- $\nu = 0$
- $CFL = 0.5$
- Number of cells $N = 1000$
- Number of particles per cell = 100

Result The result of the above simulation setup is presented in Figure 22. The x-axis represents the position and the y-axis represents the function value. The blue line marks the initial condition. The red line marks the

solution obtained by method of characteristics. And the yellow line marks the solution obtained by the stochastic particle-grid approach. We see that the stochastic result is in good agreement with the exact solution.

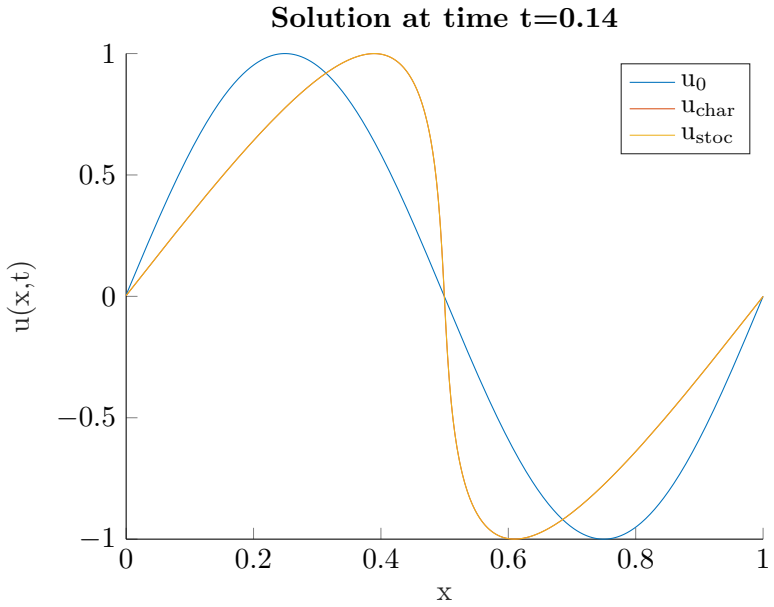

Figure 22: Particle-grid: Nonlinear ic - Inviscid - Sample solution

5.1.3 PARTICLE-GRID: NONLINEAR IC - INVISCID - GRID CONVERGENCE

Model Problem We consider the following inviscid Burgers equation with nonlinear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = 0$$
$$u(x, 0) = sin(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- periodic boundary condition
- $\nu = 0$
- $CFL = 0.5$
- Number of particles per cell = 100

Result The result of the above simulation setup is presented in Figure 23. The x-axis represents the logarithmic value of grid points and the y-axis represents the logarithmic value of $l2$ norm of error. In this test, we gradually refine our grid finely while keeping the CFL condition and the number of particles per cell constant. For each grid setup N we plot the error value. We see that the order of convergence is of some algebraic order.

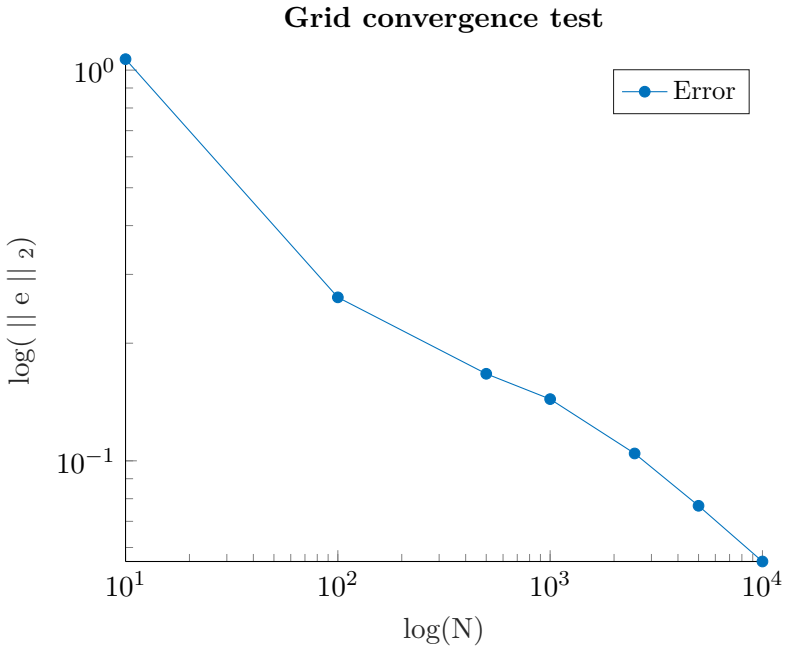

Figure 23: Particle-grid: Nonlinear ic - Inviscid - Grid convergence

5.1.4 PARTICLE-GRID: NONLINEAR IC - INVISCID - PARTICLE CONVERGENCE

Model Problem We consider the inviscid Burgers equation with nonlinear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = 0$$
$$u(x, 0) = sin(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- periodic boundary condition
- $\nu = 0$

- $CFL = 0.5$
- Number of cells $N = 1000$

Result The result of the above simulation setup is presented in Figure 24. The x-axis represents the logarithmic value of number of particles per cell and the y-axis represents the logarithmic value of $l2$ norm of error. In this test, we fix the grid parameters such as N, CFL, and final time. We increase the number of particles per cell gradually. For each N_p per cell we plot the error value. We see that the error value oscillates with number of particles. The reason for this behaviour is as follows: in the inviscid case, the introduction of more particles in the system does not have any effect on the accuracy of solution. In later section, in the viscous case, we will see that there is a correlation between number of particles per cell and the error value.

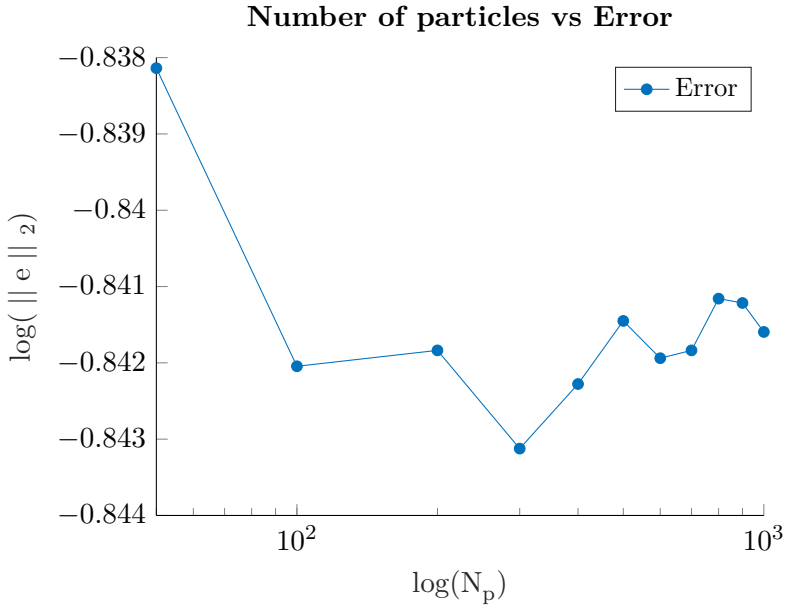

Figure 24: Particle-grid: Nonlinear ic - inviscid - particles convergence

5.1.5 PARTICLE-KERNEL: LINEAR IC - INVISCID - SAMPLE SOLUTION

Model Problem We consider the inviscid Burgers equation with linear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = 0$$

$$u(x,0) = x$$

Simulation data The following parameters are used in the simulation to obtain the results.

- free boundary condition
- $\nu = 0$
- $CFL = 0.5$
- Number of cells $N = 1000$
- Number of particles per cell = 1000

Result The result of the above simulation setup is presented in Figure 25. The x-axis represents the position and the y-axis represents the function value. The blue line marks the initial condition. The red line marks the solution obtained by method of characteristics. And the yellow line marks the solution obtained by the stochastic particle-kernel approach. We see that the stochastic result matches very accurately with the exact solution.

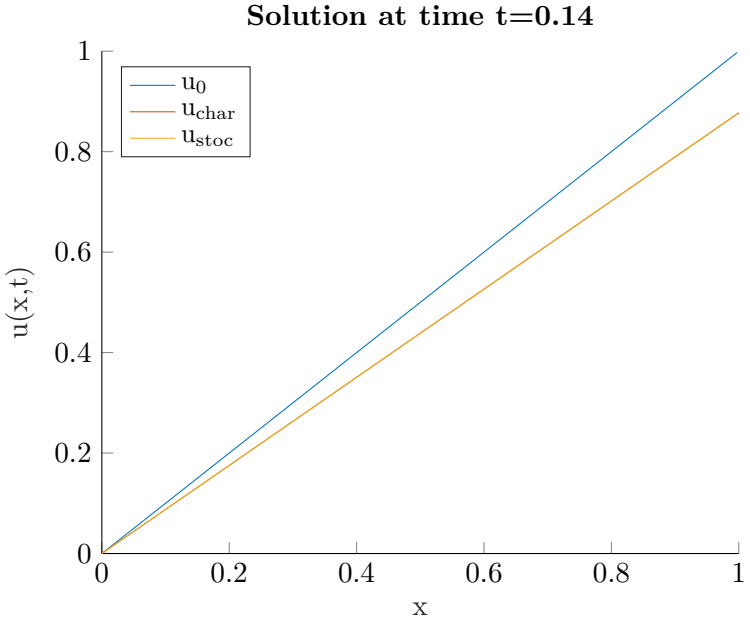

Figure 25: Particle-kernel: Linear ic - Inviscid - Sample solution

5.1.6 PARTICLE-KERNEL: NONLINEAR IC - INVISCID - SAMPLE SOLUTION

Model Problem We consider the inviscid Burgers equation with nonlinear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = 0$$
$$u(x, 0) = sin(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- periodic boundary condition
- $\nu = 0$
- $CFL = 0.5$
- Number of cells $N = 1000$
- Number of particles per cell = 1000

Result The result of the above simulation setup is presented in Figure 26. The x-axis represents the position and the y-axis represents the function value. The blue line marks the initial condition. The red line marks the solution obtained by method of characteristics. And the yellow line marks the solution obtained by the stochastic particle-kernel approach. We see that the stochastic result matches very accurately with the exact solution.

5.1.7 CHALLENGES IN THE PARTICLE-KERNEL APPROACH FOR BURGERS EQUATION

While the particle-kernel approach works well in the inviscid case, there are some challenges associated with this method that should be addressed in this section.

- With this approach we can calculate the velocity only on the particles' position. We could not calculate the velocity on any desired point in the domain.
- While this algorithm works for inviscid case well, when viscosity is added to the system, the current particle-kernel algorithm fails to produce satisfactory results.

Therefore we did not perform any convergence studies on this algorithm.

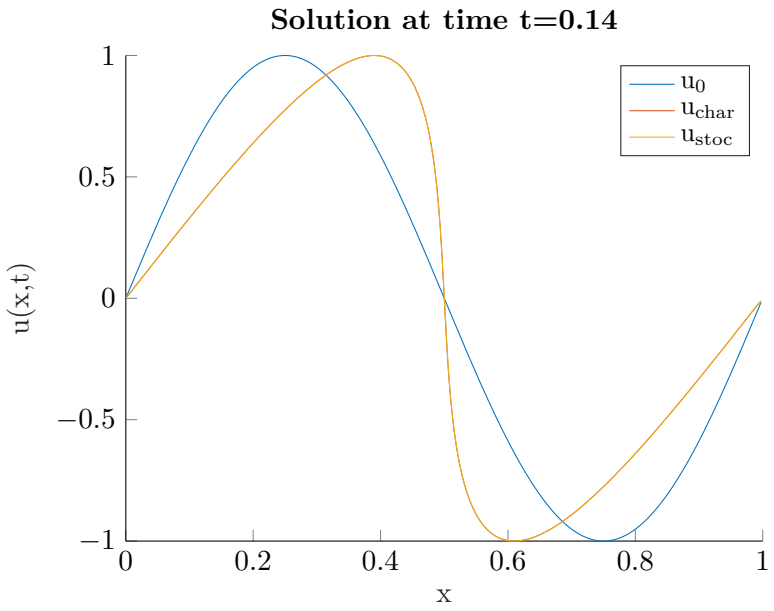

Figure 26: Particle-kernel: Nonlinear ic - Inviscid - Sample solution

5.2 VISCOUS BURGERS EQUATIONS

In this section we discuss the results of the one-dimensional viscous Burgers equation.

5.2.1 PARTICLE-GRID: NONLINEAR IC - VISCOUS - SAMPLE SOLUTION

Model Problem We consider the viscous Burgers equation with nonlinear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu \Delta u = 0$$
$$u(x, 0) = sin(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- periodic boundary condition
- $\nu = 0.01$
- $CFL = 0.5$
- Number of cells $N = 10000$
- Number of particles per cell = 100

Result The result of the above simulation setup is presented in Figure 27. The x-axis represents the position and the y-axis represents the function value. The blue line marks the initial condition. And the red line marks the solution obtained by the stochastic particle-grid approach. As we do not have an exact solution for the viscous Burgers equation, we did not plot it here. In the following we present the convergence study of this algorithm.

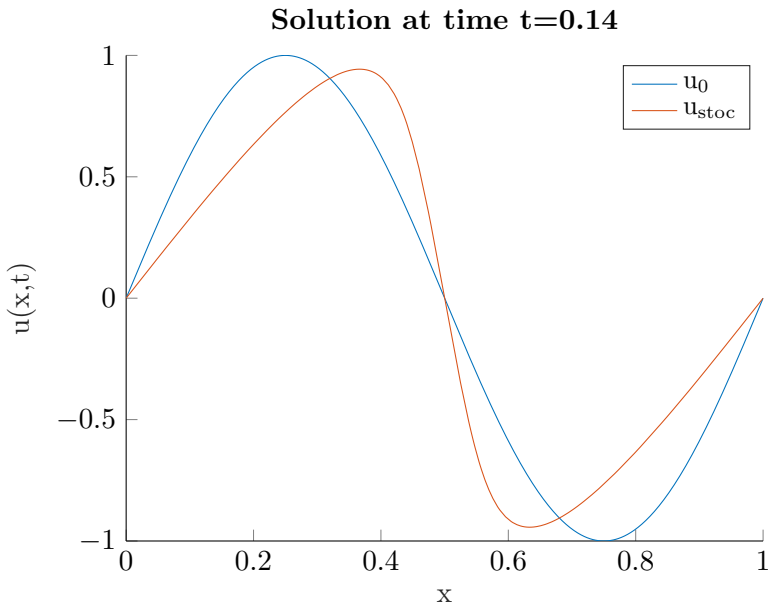

Figure 27: Particle-grid: Nonlinear ic - Viscous - Sample solution

5.2.2 PARTICLE-GRID: NONLINEAR IC - VISCOUS - GRID CONVERGENCE

Model Problem We consider the viscous Burgers equation with nonlinear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu \Delta u = 0$$
$$u(x, 0) = sin(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- periodic boundary condition
- $\nu = 0.01$
- $CFL = 0.5$
- Number of particles per cell = 100

Result The result of the above simulation setup is presented in Figure 28. The x-axis represents the logarithmic value of grid points and the y-axis represents the logarithmic value of $l2$ norm of error. The error value was calculated against the solution of the finest grid in the simulation. In this test, we gradually refine our grid finely while keeping the CFL condition and the number of particles per cell constant. For each grid setup N we plot the error value. We see that the order of convergence is of some algebraic order.

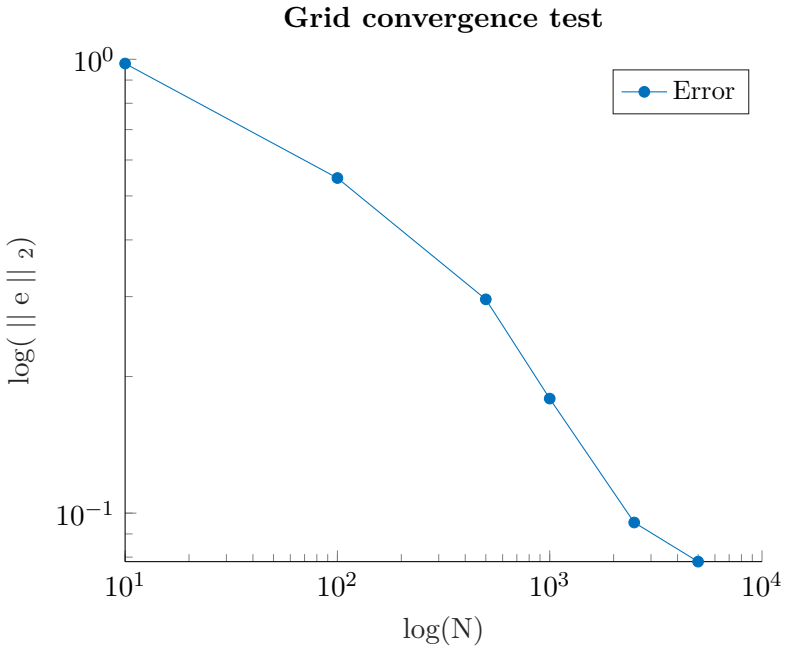

Figure 28: Particle-grid: Nonlinear ic - Viscous - Grid convergence

5.2.3 PARTICLE-GRID: NONLINEAR IC - VISCOUS - PARTICLE CONVERGENCE

Model Problem We consider the viscous Burgers equation with nonlinear initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u - \nu \Delta u = 0$$
$$u(x, 0) = sin(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- periodic boundary condition

- $\nu = 0.01$
- $CFL = 0.5$
- Number of cells $N = 1000$

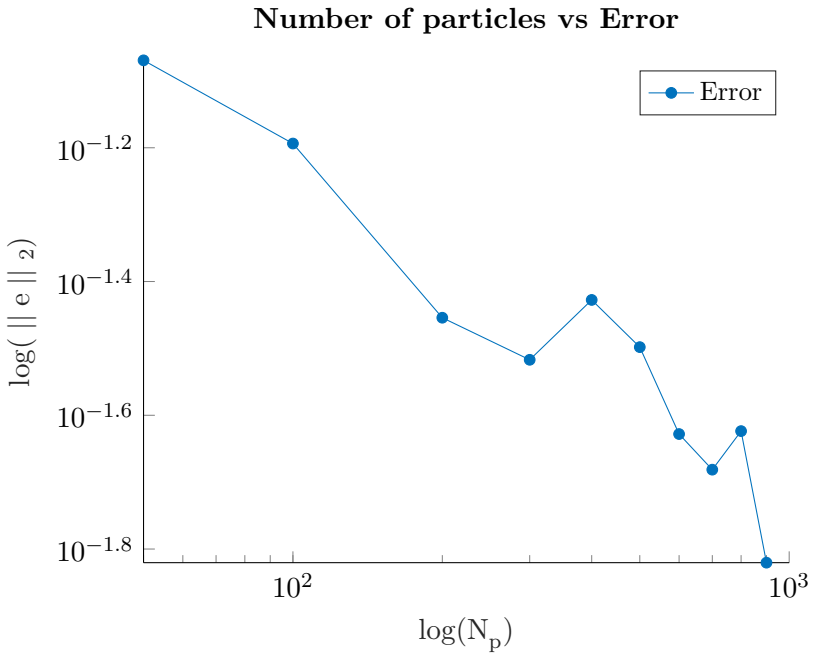

Figure 29: Particle-grid: Nonlinear ic - Viscous - Particle convergence

Result The result of the above simulation setup is presented in Figure 29. The x-axis represents the logarithmic value of number of particles per cell and the y-axis represents the logarithmic value of $l2$ norm of error. In this test, we fix the grid parameters such as N, CFL, and final time. We increase the number of particles per cell gradually. For each N_p per cell we plot the error value. Unlike the inviscid case, here we notice that the introduction more particles in the simulation does have an effect on the accuracy of the solution. The error decays with the introduction of more particles. We also note here the oscillations in the middle section of the graph. That is due to the effect of randomness. In order to obtain more accurate results, one can repeat each simulation several times and plot the average of those results.

5.3 INCOMPRESSIBLE NAVIER-STOKES EQUATION

In this section, the results of two-dimensional incompressible Navier-Stokes equation are presented.

5.3.1 PARTICLE-GRID: NAVIER-STOKES - SAMPLE SOLUTION

Model Problem We consider the incompressible Navier-Stokes equation with following initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u + \nabla p = \nu \Delta u$$
$$u_0(x, y) = sin(2\pi y) + cos(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- free boundary condition
- $\nu = 0.01$
- $CFL = 0.5$
- Number of cells $N = 400$
- Number of particles per cell = 100

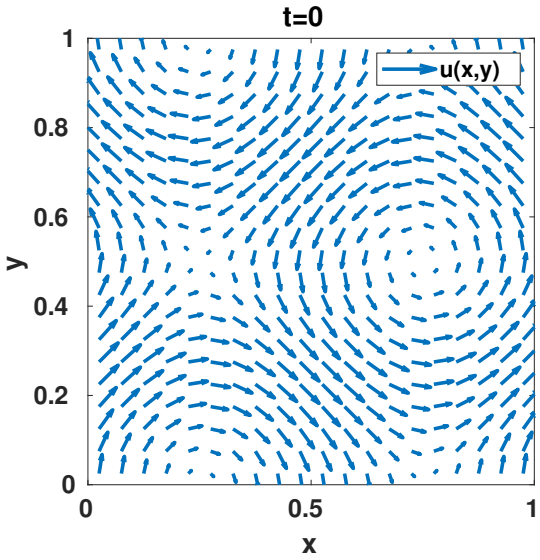

Figure 30: Particle-grid: Navier-Stokes - Initial velocity profile

Result The particle-grid approach is used to solve the above simulation setup. And the results of the simulation are presented as follows: We observe in Figure 30 the initial velocity profile at time $t = 0$ and in Figure 31 we see the velocity profile after 5 time steps.

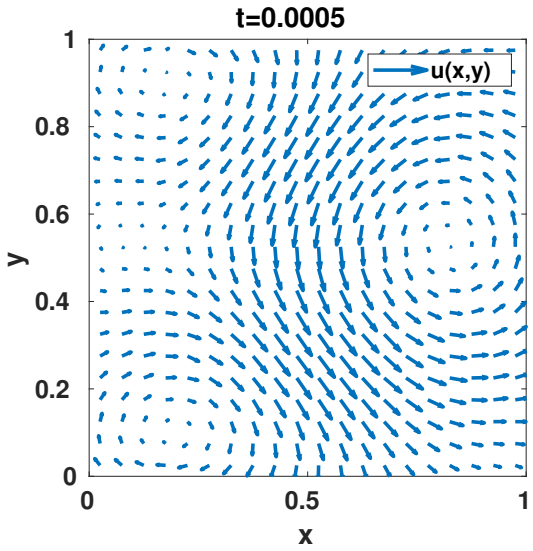

Figure 31: Particle-grid: Navier-Stokes - Velocity profile at time-step 5

5.3.2 PARTICLE-GRID: NAVIER-STOKES - GRID CONVERGENCE

Model Problem We consider the incompressible Navier-Stokes equation with following initial condition as our problem statement.

$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u + \nabla p = \nu \Delta u$$
$$u_0(x, y) = sin(2\pi y) + cos(2\pi x)$$

Simulation data The following parameters are used in the simulation to obtain the results.

- free boundary condition
- $\nu = 0.01$
- $CFL = 0.5$
- Number of particles per cell = 100

Result The result of the above simulation setup is presented in Figure 32. The x-axis represents the logarithmic value of grid points and the y-axis represents the logarithmic value of $l2$ norm of error. The error value was calculated against the solution of the finest grid in the simulation. In this test, we gradually refine our grid finely while keeping the CFL condition and the number of particles per cell constant. For each grid setup N. the error value is plotted. We see that the order of convergence is of some algebraic order.

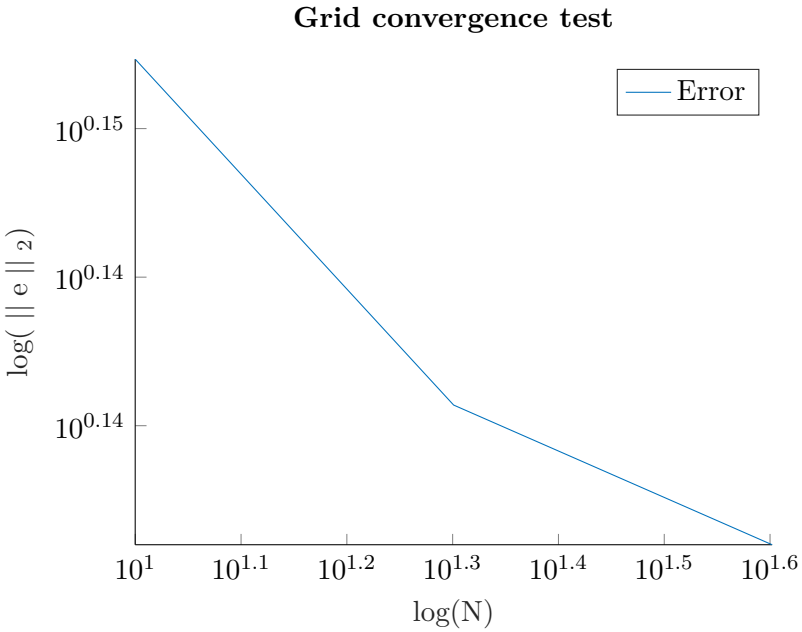

Figure 32: Particle-grid: Navier-Stokes - Grid convergence

6 CONCLUSION

Summary In the following, we recapitulate the work done in the thesis.

- Solution algorithms for Burgers equation and incompressible Navier-Stokes equation were successfully developed.
- Two approaches, particle-grid and particle-kernel approach, were used for developing the solution algorithms.
- Particle-grid algorithm was demonstrated by applying it on inviscid Burgers and viscous Burgers equations. And related convergence studies were performed for each case.
- Particle-kernel algorithm was demonstrated by applying it on inviscid Burgers equation. The challenges in the viscous case were discussed.
- Particle-grid algorithm for Navier-Stokes equation was demonstrated and related convergence study was performed.

In this thesis, we concluded that stochastic particle method is a viable option to solve particle differential equations. Especially, it is a suitable approach to solve systems with uncertainties.

Pointers to future work

- When particle-kernel algorithm was applied to solve viscous Burgers equation, the algorithm did not produce satisfactory results. The possible reasons are as follows: Kernel choice and distribution of particles. There could also be other measures that cause this unsatisfactory results. It remains to be investigated.
- The developed particle-grid algorithm for Navier-Stokes equation does not include boundary effects in the algorithm. This nontrivial task is a point of interest for the future work: inclusion of other boundary condition types.
- Once the successful development of particle-kernel approach for viscous Burgers equation is achieved, it is interesting to develop the same approach for Navier-Stokes equation and study its convergence rate.
- The development of a multiscale solver that is completely based on particle method is the ultimate goal. This work serves as a starting point for one of the many components of that multiscale solver.

BIBLIOGRAPHY

[1] Chorin A. J. , Numerical study of slightly viscous flows, Jornal of Fluid Mechanic 57, number 4, pages 785-796, 1973 .

[2] Busnello B. , A probabilistic approach to the two-dimensional Navier-Stokes equation, Ann. Probab. 27, pages 1750-1780, 1999.

[3] Joseph L. Doob, Stochastic processes, ISBN: 0-471-52369-0, 1953.

[4] Nobert Wiener, Nobert Wiener: Collected Works, Volume 1, ISBN: 9780262230707, 1976.

[5] Amir H. Delgoshaie et al, The stochastic counterpart of conservation laws with heterogeneous conductivity fields: Application to deterministic problems and uncertainty quantification, Journal of Computational Physics: X, 2019.

[6] Mireille Bossy, Comparison of a stochastic particle method and a finite volume deterministic method applied to Burgers equation, Monte Carlo Methods and Appl. Vol. 3, page 113-140, 1997.

[7] Peter Constantin and Gautam Iyer, A stochastic Lagrangian representation of the three-dimensional incompressible Navier-Stokes equations, Communications on Pure and Applied Mathematics, pages 330-345, 2008.

[8] Stephan Roberts, A particle method for a scalar advection diffusion equation, Mathematics and Computers in Simulation 32, pages 155-160, 1990.

[9] Ahmed F. Ghoniem and Frederick S. Sherman, Grid-free Simulation of Diffusion using Random Walk Methods, Journal of Computational Physics 61, pages 1-37, 1985.

[10] Smoothed Particle Hydrodynamics, J.J Monaghan, Annual Rev. Astron. Astrophys. 30, pages 543-574, 1992.

[11] Chien-cheng chang, Random Vortex Methods for Navier-Stokes equation, Journal of Computational Physics 76, pages 281-300, 1988.