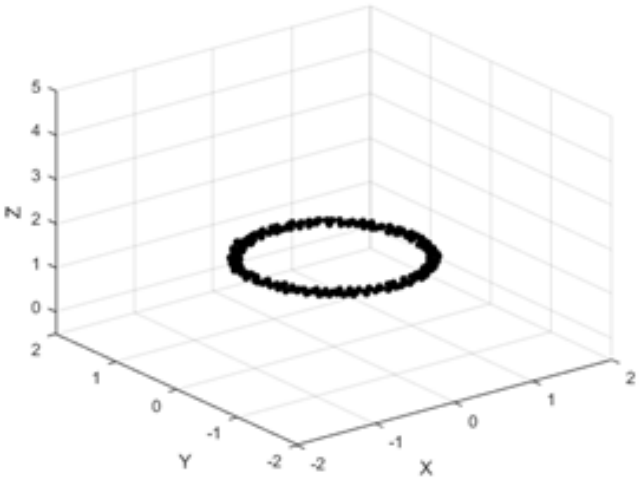
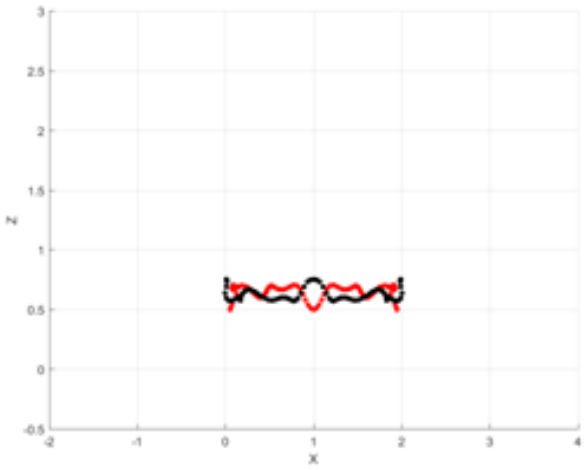
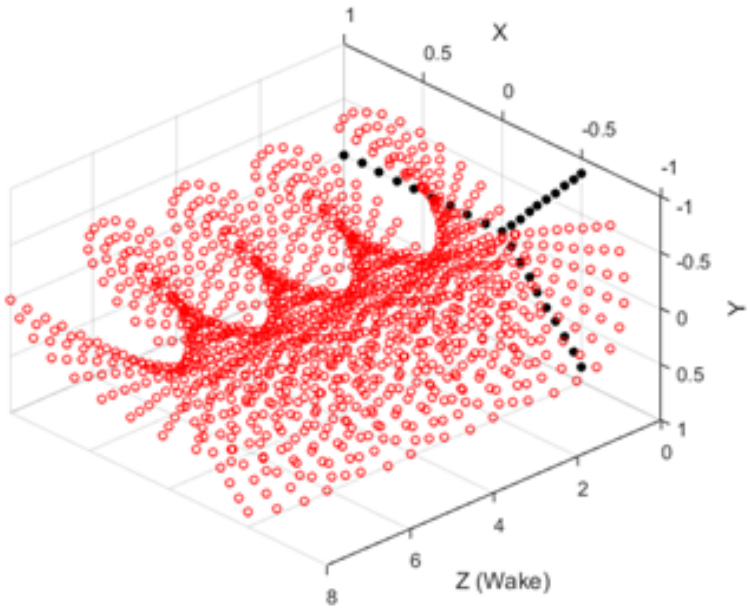




$$\frac{D\vec{\omega}}{Dt} = (\vec{\omega}.\nabla)\vec{u} + \upsilon\nabla^2\vec{\omega}$$



Implementation of a vortex particle scheme to analyze the wake of a wind turbine modeled using actuator lines

Venkatesan Seetharaman

TU Delft

Implementation of a vortex particle scheme to analyze the wake of a wind turbine modeled using actuator lines

by

Venkatesan Seetharaman

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	4503457	
Project duration:	October 1, 2017 – March 28, 2019	
Thesis committee:	Prof. dr. ir. A. H. van Zuijlen,	TU Delft, Supervisor
	Dr. dr. ir. B. W. van Oudheusden,	TU Delft, Chairperson
	Dr. dr. ir. Carlos Simao Ferreira,	TU Delft, External committee member

This thesis is confidential and cannot be made public until March 28, 2019.

TU Delft

Preface

This thesis for ths research originally stemmed from the goal of developing a wind farm solver to study the far wake of a wind turbine. During the course of the research a wind turbine wake solver has been developed in MATLAB using VPM with a Particle Strength Exchange scheme being used to simulate diffusion in the velocity-vorticity form of the Nvier-Stokes equations. As VPM involves the solution of an n-body problem, it was sought to accelerate the execution of the code using a Fast Multipole Method algorithm available as library for C++. For this purpose the available library has been validated and the errors that it generates analyzed. This research has been successful at being able to represent the wind turbine blade as an actuator line and its wake with vortex particles that have been shed from the actuator line. Through this research the method of application of particle strengths to the shed particles in the wake has been determined and the work demonstrates that the combination of actuator line and VPM to have considerably large accuracy in the region of wake up to twice the diameter of the wind turbine with respect to the results obtained from a CFD simulation published by NREL.

I am extremely grateful for the guidance I received for this work from Dr. A. H. van Zuijlen without which it would have been difficult to see the order amidst all the chaos that had arisen from my limited experience in C++ and numerical techniques.

I am also thankful for all the input that I received from Shaafi Kaja Kamaludeen which enabled me to clearly understand the concept of VPM during the very early stages of this thesis.

Many thanks to my loving parents whose support and encouragement played a major role in this work.

Venkatesan Seetharaman
Delft, March 2019

Contents

List of Figures

Nomenclature

α	Circulation of a particle obtained as a product of vorticity and volume of the particle
Γ	Circulation of velocity field
ν	Kinematic viscosity
ΩR	Angular velocity of the blade with Radius R
ω	Vorticity
ψ	Stream function
ρ	Fluid density
σ	Cut-off radius
σ_K	Core radius of vortex ring
σ_{diff}	Core radius due to diffusion
σ_{str}	Core radius due to vortex stretching
$\vec{u}$	Fluid flow velocity
g	Acceleration due to gravity
$K\times$	Biot-Savart kernel
Kq	Matrix-vector product of K (square matrix) and q
l	Length of the filament
N	Number of vortex particles
q	Column vector for source strength
R	Radius of vortex ring
t	Time
U_0	Velocity of incoming wind
U_{eff}	Effective velocity of the wind experienced by the rotating blade
vol	Volume of fluid particle
VS	Vortex stretching term

Abstract

The efficiency of a wind turbine depends largely on the wake of the upstream turbine. Seeking to contribute towards the development of a wind farm solver using a Lagrangian scheme to analyze the wake, this thesis analyses and validates a Vortex Particle Method (VPM) algorithm by simulating the behavior of vortex rings. Due to the computationally expensive nature of VPM schemes to solve n-body problems the algorithm has been validated to be possible to accelerate using an FMM library called BBFMM3D which was found to reduce computational times extensively. Using the VPM solver developed to simulate vortex rings, the simulation of a wake of a wind turbine, that was modeled using actuator lines, has been analyzed and attempted to be validated. It has been found that the VPM scheme generates considerably agreeable results of the wake with respect to other CFD simulations. This thesis has demonstrated the possibility to combine an actuator line model with a vortex particle scheme to simulate the wake. However, the accuracy of the results has been found to rely significantly on the formulation of the strengths of the vortices shed into the wake. Two formulations for this purpose have been presented with the results showing signs of improvement from one formulation to another.

1 Introduction

1.1. Motivation to use vortex particle method to solve for wind turbine wake

A wind turbine converts the extracted kinetic energy from the incoming wind into electrical energy. Wind energy development involves grouping of such wind turbines in a particular geographical area to extract the most energy from. These groups of wind turbines are called wind farms. The power output of the entire wind farm equals the sum of the power output of the individual wind turbines in the wind farm. However, the efficiency of individual wind turbines depend on the incoming fow and this largely affects the power output of the wind farm. This is because as the wind turbine extracts the kinetic energy from the upstream wind, the downstream wake is no longer uniform and becomes turbulent before it interferes with another wind turbine. The output of the wind turbine downstream now largely depends on the wake of the upstream wind turbine. For this purpose it is important to analyze the wake of a wind turbine to be able to determine its effects on the wind turbines downstream and thereby help in determining the power output of a wind farm efficiently. Wind turbine wakes have long been of concern for a wind farm design. This thesis derives its motivation from the recent developments in wind farm design as humanity moves towards an era of renewable energy.

To entirely address the problem of the power output of a wind farm, one would need to have a valid wake analysis tool which could then be expanded into analyzing the effect of wakes on the interfering wind turbines by simulating a wind farm. Most of the works involve an actuator disc model for the wind turbine or a panel discretization scheme for the wind turbine blades to generate the wake. However, in recent years much research has been carried out in the field of Lagrangian vortex methods that allow us to track individual particles in the wake. This demands a large computational capacity as it would potentially involve quite a large number of particles in the wake and a long time to develop a complete wind farm solver. Due to this, this thesis is restricted to studying the feasibility and validity of an FMM package called BBFMM3D and attempting to create a wake analysis tool using vortex particle method aided by BBFMM3D to accelerate the computation when a large number of particles come into play in the wake.

1.2. Advantages of using vorticity particles

The velocity-vorticity equation in 1.3 is a Lagrangian equation describing the evolution of the vorticity of fluid particles over time. The numerical solution to velocity-vorticity equation has been under research since as early as the 1930s. In order to obtain a sufficiently accurate results using the vortex methods one would need a very large number of particles, of the order of 10^5, in the fluid domain to simulate a flow. Therefore, due to lack in computational capabilities at the time not much research has been done using vortex methods. In recent times, however, there has been significant development in the field of vortex methods that were developed to solve the vorticity equation numerically. The literature survey on the history of developments in the field is presented in chapter 2 of this thesis.

A salient feature of the Lagrangian vortex particle method is that it is mesh-free. The continuum of the fluid

domain to be simulated is discretized into particles carrying vorticity. The strength of the vorticity field of particles can be considered to form a field in space and time. It is the evolution of this field that the vortex methods deal with and this makes it possible to track every particle in the domain of the fluid flow, i.e., the computation involved in this method is localized. This Lagrangian description of vorticity particles has many advantages over other conventional CFD techniques such as the finite difference method. The vortex particle method is free of numerical dissipation which is a cause of errors in the grid-based methods. Also, the number of particles is easily adapted to the complexity of the flow and this method is also suitable for both internal and external flows.

Computational fluid dynamics deals with numerically solving the Navier-Stokes equations 1.1-1.2, that govern the fluid flow, to simulate fluid motion. In the below equations, velocity $\vec{u}$ is the unknown quantity. In the below equations, p is the pressure, ρ is the density of the fluid and ν is the kinematic viscosity.

$$\nabla \cdot \vec{u} = 0 \tag{1.1}$$

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = g - \frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{u} \tag{1.2}$$

However, there is another form of the Navier-Stokes equations which also describes fluid motion. Taking the curl of the Navier-Stokes equation leaves us with the velocity-vorticity equation 1.3 and is an alternate means to describe fluid motion using the vorticity in the fluid particles.

$$\frac{D\vec{\omega}}{dt} = (\vec{\omega} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{\omega} \tag{1.3}$$

This equation helps us simulate fluid flow by describing the motion of individual particles. In equation 1.3, $\vec{\omega}$ represents the fluid particle's vorticity and is an unknown quantity. To understand the differences between the regular Navier-Stokes equation and the velocity-vorticity form let us take a step back to understand the approaches through which fluid motion can be described.

1.3. Lagrangian approach

In the Lagrangian perspective the flow is described for a fixed identity of particles. If $\vec{X}(\vec{X}, t)$ represented the position vector of a particle and $\vec{X} = \vec{X}_0$ was the position of the particle at time $t = t_0$ as represented in equation 1.4, then the velocity of the particle is given by 1.5.

$$\vec{X}_0 = \vec{X}(\vec{X}_0, t_0) \tag{1.4}$$

$$\vec{V}(\vec{X}, t) = \frac{d\vec{X}(\vec{X}, t)}{dt} \tag{1.5}$$

1.4. Eulerian approach

In the Eulerian description the fluid properties are specified at a fixed location for varying time. If $\vec{v}(\vec{x}, t)$ represented the velocity of particles as a function of position $\vec{x}$ and time t, the Eulerian description for velocity $\vec{v}$ is given by equatio 1.6.

$$\vec{v}(\vec{x} = \vec{X}_0, t) = \frac{d\vec{x}(\vec{x} = \vec{X}_0, t)}{dt} \tag{1.6}$$

The descriptions can be related using the material derivative which is defined, as follows, for any quantity $\vec{f}$ in equation 1.7

$$\frac{D\vec{f}}{Dt} = \frac{\partial \vec{f}}{\partial t} + (\vec{v} \cdot \nabla)\vec{f} \tag{1.7}$$

In the above expression the term on the left hand side represents the Lagrangian variation and the first term

on the right hand side is the Eulerian variation (evaluated at a fixed position) of $\vec{f}$ and the second term is the convective change of $\vec{f}$.
It can be seen from equation 1.3 that the rate of change of vorticity is represented in its Lagrangian form. This form of the equation helps us track individual particles in the fluid domain as their vorticities evolve over time due to convection and diffusion.

1.5. Wake models

This section briefly outlines the two analytical wake models that are in use today. These wake models help us in describing the physics of the wake and predict the wind velocity downstream at the chosen distances from the blade.

1.5.1. Jensen's model

The Jensen model was the first analytical wake model developed by N.O. Jensen in the year 1983 and was improved upon by Katic [26]. This model is based on the mass and momentum conservation and on the assumption that the wake downstream of the blade expands linearly on account of losing energy at the blade. This model is still in use today due to its simplicity in computing the velocity profile and validation.
Recently, this model has been updated with the methods for partial wake interaction, yawed flow and correction for wind direction changes.
Combining the conservation of mass and applying a linear variation to the radius of the wake as given in equation 1.8, this model predicts the wake velocity to be

$$r = kz + r_0 \tag{1.8}$$

$$U = U_\infty[1 - 2a(\frac{r_0}{r_0})^2] \tag{1.9}$$

where U is the velocity of the wake at any section, z, downstream, a is the induction factor, k is the wake decay coefficient, and r_0 is the rotor radius.

1.5.2. Ainslie's model

To solve the wind velocity components, the Navier-Stokes equation with the unknown Reynolds-averaged shear stress term has to be solved with the continuity equation. Ainslie's model expresses this shear stress term by the eddy viscosity. Using an empirical equation, this eddy viscosity is expressed in terms of the wind velocity component. Thus, Ainslie's model is also referred to as the Eddy viscosity model owing to its methods of computing the viscous stresses that are modelled after eddy viscosity.

This wake model developed by Ainslie assumes that the solution for velocity components starts from the downstream flow region where the pressure gradient is no longer dominant. The essence of this model is that it assumes a gaussian profile for the near wake and the atmospheric turbulence is used to calculate the initial velocity.

1.6. Fidelity of Wind Farm solvers

The power loss of a wind turbine downstream has been noted to vary between 8% for an onshore wind farm and 12% for an offshore wind farm. To predict the power output of a wind farm it is important to accurately model the wake and its interaction to study the effects on the wind turbine's performance. But accuracy always trades off with computational power. It is not always required to model a wind farm with all details of physics involved in the model. The fidelity of the solver describes how detailed a result the solver can produce based on the physical assumptions that are coded in the solver. This section explains the three fidelity of wind farm solvers that are in practice. This is thesis focuses on creating a medium fidelity solver.

1.6.1. High Fidelity solvers

High fidelity models are those that include a high level of details and very few assumptions. These models include blades as solid boundaries. High fidelity models are helpful in the sense that highly accurate force distributions can be obtained as the model involves few assumptions. This reflects in the accurate wake

evolution. But for such hi fidelity models it is required that the domain consists of highly refined meshes and hence this is computationally expensive.

1.6.2. Medium fidelity solvers

Navier-Stokes equations form the basis for the wake physics but the blade model may not be solid. Instead, an actuator disc may be used to represent the blade and such assumptions are helpful in analyzing the far wake in the simulations.

1.6.3. Low-fidelity solvers

In such models the order of physics is reduced to the extent that gives acceptable accuracy and at an affordable computational cost. For example, the Jensen-Katic model, the wake deficit and the kinetic energy deficit of the interacting wakes is considered to be the sum of the kinetic energy deficit of the individual wakes

1.7. Scope of this thesis

In this thesis focus in laid upon the development of a wind turbine wake solver using vorticity particles (vortex particle method) and validating an FMM library that may be integrated with the solver to accelerate the evolution of the wake (n-body problem) using the velocity-vorticity equation. In this regard, this chapter outlined the advantages of using vorticity particles to simulate the flow in the wake and the various wake models that can be implemented to predict the wake velocity profiles. Chapter 2 describes the vortex method briefly before moving on to explain the applications of VPM in aerodynamics and wind turbine in the past. Chapter 3 describes the vortex particle method that is used in this work followed by a description and validation of FMM tool BBFMM3D in chapter 4. Chapter 5 validates the code written to simulate vortex rings. Chapter 6 and 7 present the results of the wind turbine wake solver developed using MATLAB and C++ respectively.

2

Literature survey on vortex particle method and its applications

2.1. Brief description of vortex methods

In the following sections we briefly describe the vortex methods in general, the terms involved in the velocity-vorticity equation, the diffusion methods developed in history and a literature survey on the applications of vortex particle methods in aerodynamics and wind turbine.

Let us begin with the definition of vorticity. For a velocity field $\vec{u}(\vec{x}, t)$ an angular velocity $\vec{\Omega}$ can be defined as $\vec{\Omega} = \frac{1}{2}(\nabla \times \vec{u}$. A vector that is twice the angular velocity vector is defined as the vorticity, $\vec{\omega} = \nabla \times \vec{u}$. It is made to be twice as large to avoid the discrepancy of a factor $\frac{1}{2}$.

As in chapter 1, equation 1.3 describes fluid flow using the velocity-vorticity equation in a vorticity field. This equation describes the vorticity field to be changing as the vorticity particles are transported along with stretching and diffusion.

To discretize the vorticity field we approximate it as the linear combination of vorticities of the particles in the flow as represented in 2.1.

$$\vec{\omega} = \sum_{i=1}^{N} \zeta(\vec{x} - \vec{x}_i)\Gamma_i \tag{2.1}$$

In the above equation Γ is the vorticity strength multiplied by the volume of the particle and is approximately $\vec{\omega}_i V_i$. ζ may be a Gaussian distribution but we use a function as described by Winckelmans [35].
The induced velocity is obtained from the stream function ψ that satisfies $\nabla^2\psi = -\vec{\omega}$ and the velcity is obtained by convoluting K and $\vec{\omega}$, as in 2.2, where $K\times$ is the Biot-Savart kernel.

$$\vec{u} = K \times \vec{\omega} \tag{2.2}$$

2.2. Vortex stretching

Vortex stretching is expressed by the first term on the right hand side of equation 1.3. This term is responsible for the expansion and contraction of a vortex particle, thereby, increasing or decrasing the magnitude of vorticity associated with the particle. This is depicted in the figure 2.1. Considering a cylindrical vortex element with an initial vorticity ω_1, the vorticity increases to ω_2 when the filament gets stretched on account of cnservation of angular momentum of the spinning fluid particle or filament.

In 2D flow, the stretching term goes to zero as the vorticity and velocity vectors are perpendicular. In 3D, stretching causes a change in vorticity at every time-step. This vorticity stretching term requires that the vorticity field is divergence free.

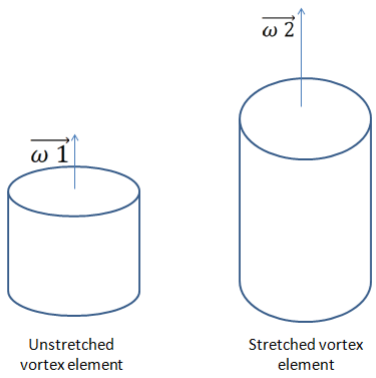


Figure 2.1: Depiction of vortex stretching for a cylindrical fluid element

2.3. Diffusion schemes

2.3.1. Random walk method

Chorin, in 1973, introduced the Random walk method to solve for the diffusion term in the velocity-vorticity equation. This was a very simple approach to simulate diffusion. The idea was to follow the viscous splitting algorithm and implement the random walk method using the vorticity field established in the convection step of the algorithm. The particles in the domain are made to undergo a Brownian movement to simulate diffusion. To simulate the Brownian movement the particles are made to undergo random position updates at every time step, δt, according to equation 2.3, where p represents the particle index and n represents the current time step. ξ represents a random number generated following a Gaussian distribution as given in 2.4, where d represents the dimension, ϵ represents the variance and is equal to $2\nu\delta t$

$$\vec{x}_p^{n+1} = \vec{x}_p^n + \xi_p^n \tag{2.3}$$

$$\xi = \frac{1}{(\sqrt{2\pi\epsilon})^d} e^{\frac{-x^2}{2\epsilon}} \tag{2.4}$$

Although this method did approximate the diffusion process as expected, it did not account for any changes in the magnitude of the vorticity due to diffusion. Hence this method did not prove to be a valid option to simulate diffusion.

2.3.2. Core spreading method

in 1973 Kuwahara and Takami [19] studied the 2D vortex motion using a method known as core spreading method. Greengard [14] discredited the method on account of mathematical inaccuracies. So until around 1990 the only methods that were in use, to simulate diffusion, was the random-walk method and the core spreading method and Winckelman's particle strength exchange in 1989. Core spreading method gained popularity when Rossi provided a correction to the original method. Leonard [20] presents a Lagrangian scheme for the core spreading method to exactly solve the diffusion equation.

In 3D the core radius is changed on account of stretching and diffusion separately.

$$\frac{d\vec{\omega}}{dt} = (\vec{\omega}\cdot\nabla)\vec{u} \tag{2.5}$$

$$\frac{dl}{dt} = \frac{l_t}{\vec{\omega}_t}\frac{d\vec{\omega}}{dt} \tag{2.6}$$

$$\frac{d\sigma_{str}}{dt} = -\frac{\sigma}{2l_t}\frac{dl}{dt} \tag{2.7}$$

$$\frac{d\sigma_{diff}}{dt} = -\frac{c^2\nu}{2\sigma_t} \tag{2.8}$$

$$l_{t+\Delta t} = l_t + \frac{dl}{dt}\Delta t \tag{2.9}$$

$$\omega_{t+\Delta t} = \omega_t \frac{(\sigma_{t+\Delta t})}{\sigma_t} \tag{2.10}$$

A new blob now replaces the vortex particle.

2.3.3. Particle strength exchange

This method approximates diffusion by distributing circulation among particles in the domain. It is based on the assumption that the circulation of the particles depends upon the location of all the other particles. Degond and Mas-Gallic [10] proposed the idea of approximating the laplacian using an integral operator for prescribed conditions of the smoothing function. The approximation is as given in equation 2.11

$$\nabla^2\vec{\omega} \approx \frac{2}{\sigma^2}\int \zeta_\sigma(\vec{x}-\vec{x}')[\vec{\omega}(\vec{x})-\vec{\omega}(\vec{x}')]d\vec{x}' \tag{2.11}$$

The derivation of the final discretized formulation of the evolution equations is presented in chapter 3.

2.4. Recent applications of vortex particle methods in aerodynamics

Vortex particle method has been used in a versatile domain of applications to simulate the physics of fluid flow. A hybrid method involving the Navier-Stokes equation in the velocity form and the velocity-vorticity equation in a mesh-free domain, the vortex particle method has been used to simulate smoke, fire and explosions in the special effects industry [28]. Many such research have been carried out to improve the simulation of high reynolds number flows using VPM. In the field of aerodynamics, most of the applications use a combination of discretizing the geometry in the flow domain with panels before converting the panels into vortex particles. The following paragraphs explain this.

2.4.1. Simulating aircraft wakes

The flow around an aircraft simulated using a combination of pFFT, FMT and vortex particle method has been presented in [34]. pFFT and FMT are used to accelerate the computational speed during the phase of evaluating the velocity potential on the aircraft. The trailing vorticity is then computed using a Finite Multipole vortex particle method to accelerate the computation when a large number of particles are involved in the domain. The figure 2.2 shows the results from simulation as presented in [34].

2.4.2. Propeller-airframe interaction

Calabretta [5] has developed a code to simulate the propeller-airframe interaction by combining the methods of panels and particles in the wake of the propeller. In his work, the propeller is modelled using actuator disc model using vorticity particles. This is then integrated with a panel code called APAME to study the communication between particles and the panels.

2.5. Wind turbine wake simulation strategies with vortex particles

This chapter presents the strategies that have been implemented so far by researchers to simulate the wake of a propeller or a wind turbine. Since the functioning of the rotor/propeller is physically the opposite of that of a wind turbine the methods presented in this chapter have been useful in understanding the implementation of vortex particle methods in the current work of this thesis involving wind turbines. The following sections present four wake simulation strategies that are in use today which are then followed by an assessment of the methods.

2.5.1. Method I: Using GENUVP to study rotor aerodynamics

Voutsinas et al studied the rotor aerodynamics and aeroacoustics with the help of GENeralized Unsteady Vortex particle code [25]. Their work makes use of the Helmholtz decomposition which states that

$$\vec{u}(\vec{x},t) = \vec{u}_{ext}(\vec{x},t) + \vec{u}_{solid} + \vec{u}_{wake}(\vec{x},t)$$

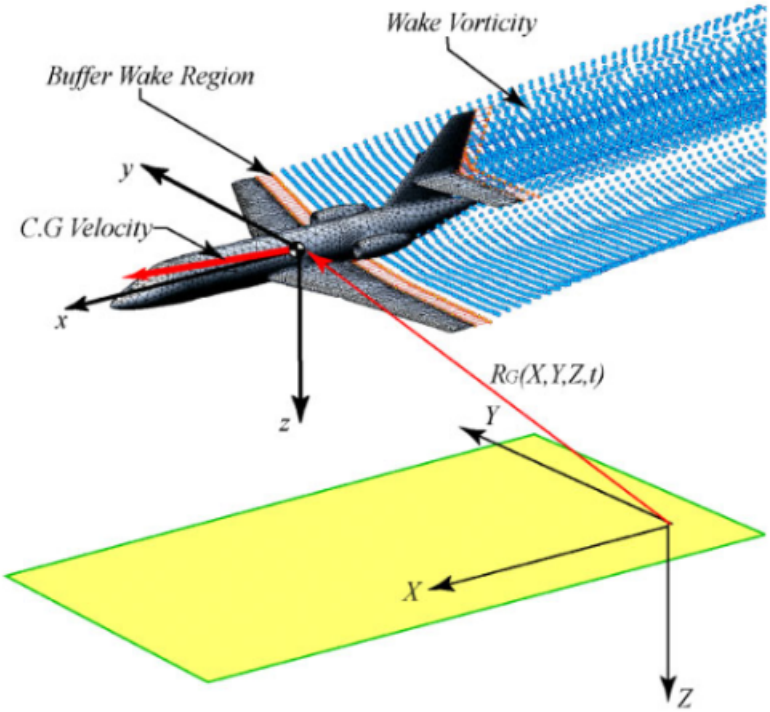


Figure 2.2: Vortex particle wake generated using pFFT in FastAero3D,figure taken from [34]

In this decomposition the second term is modeled using a panel method and the third term with $\vec{u}_{wake}$ is obtained from the Biot Savart law:

$$\vec{u}_{wake} = \int_D \frac{\omega \times (\vec{x}_0 - \vec{x})}{4\pi|\vec{x}_0 - \vec{x}|^3} dD \tag{2.12}$$

where D represents the domain where the integral is valid.

In this method the panels are modeled with dipole distributions and the trailing and tip edge are emitted at every time step. The strip elements are transformed into vortex particles by integrating each near wake dipole element.

2.5.2. Method II: Discretizing the blade into aerodynamic segments

This method is as explained in the work of He et al where they have implemented a viscous VPM taking into consideration the stretching and diffusion terms [15]. In this method the airloads acting on each aerodynamic segment is calculated with the help of the angle of attack at that segment, the mach number and dynamic pressure. Using the Kutta-Jukowski theorem and the computed airloads the blade bound circulation is calculated and is assumed to be constant over each blade segment. The vorticity source that is shed into the wake after being created at the blade segments is given by:

$$\gamma_w = -\frac{d\Gamma_b}{dt} + \mathbf{v}_b \nabla \cdot \Gamma_b$$

2.5.3. Method III: Doublets at the centre of trailing edge segments

In this method [38] the rotor surface is discretized into N panels. A constant doublet distribution is assumed. The below equation is satisfied at the centre of every panel(with the no slip condition) giving N linear algebraic equations.

$$\iint \mu\vec{n} \cdot \nabla\left(\frac{x_0 - x}{|x_0 - x|^3} \cdot \vec{n}\right) d\vec{x} = 4\pi(U + U_i) \cdot \vec{n}$$

where μ is the doublet strength, U is the freestream velocity and U_i is the induced velocity. The above equation is solved for μ to determine the doublet distribution which is then converted to its vorticity equivalent at the centre of every trailing edge segment that is emitted.

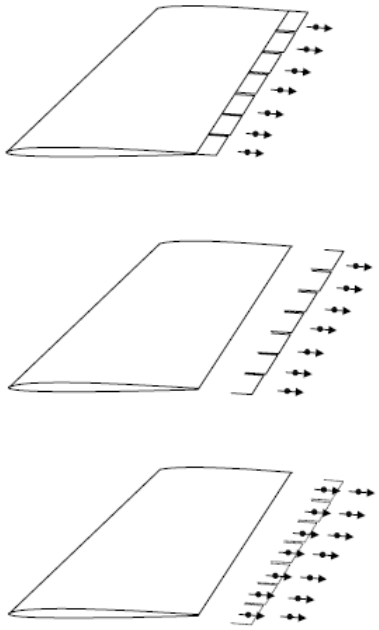


Figure 2.3: Conversion of trailing edge strips into vortex particles (Figure taken from [25])

2.5.4. Method IV: Panel and VPM for modeling stationary propeller wake wash

In the work done by Evan H Martin [22] the wake of a propeller has be modeled using the vortex particle method while the aerodynamics of the blade itself is modeled using a panel method by discretizing the geometry of the blade into panels of vortices. As these panels are shed into the wake they are converted into vortex particles, as shown in the figure. For this purpose the velocity induced by the vortex panel is considered to be equivalent to that induced by a ring vortex around the edge of the panel. Each side of the panel is considered separately and is replaced by a vortex particle whose strength is proportional to the length of the panel. This can be noted from the figure

2.6. Conclusion

All the methods described above make use of the idea of converting the panels or aerodynamic segments of the blade into vortex particles in the wake flow. It is the method of conversion that is important to be analyzed in order to be able to assess the method that have been implemented in the past.

Method I converts the panels with dipole distributions into vortex particles when they are shed into the wake by integrating each near wake dipole element. Integrating the dipole elements to convert them into vortex particles can be more or less time consuming and hence computationally more or less expensive depending on the number of elements that need to be converted. While fewer panels can be converted into vortex particles in less time, this will be done at the cost of accuracy.

Method II employs the conservation of vorticity to generate a vorticity source on each blade segment from the value of bound circulation obtained by solving the Kutta-Jukowski's equation. In this method a constant value of the bound circulation was assumed instead of considering a variation in the spanwise or azimuthal directions because the number of aerodynamic blade segments were very large. Although this might provide an agreeable solution with large number of segments, the assumption of a varying bound circulation in the spanwise direction is needed in order to obtain accurate results for a rotor because the velocity of the flow at different radial positions of the rotor blade varies in proportion to the radius. It is, therefore, essential that the bound circulation be considered a variable in the spanwise direction.

Method III employs a constant doublet distribution across across the panels on the blade which is then converted to its vorticity equivalent on the trailing edge segments. However, it is not known how a variation of

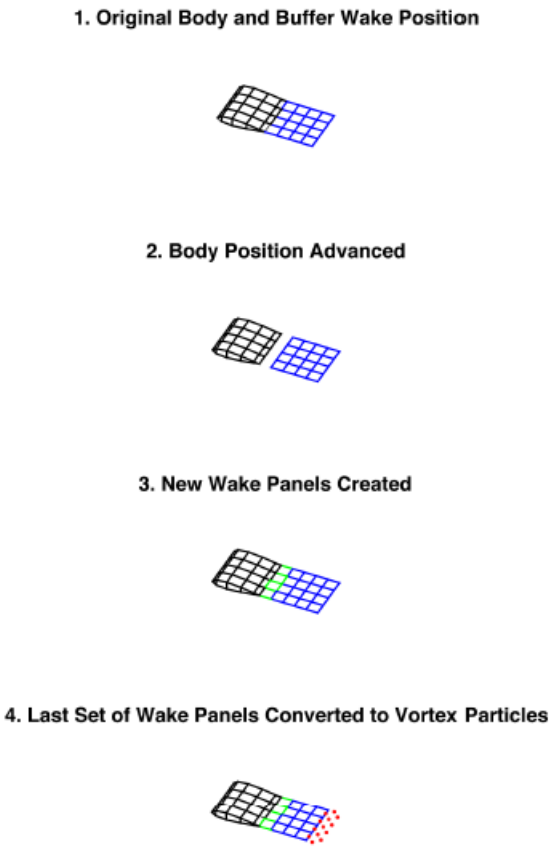


Figure 2.4: Conversion of panels into vortex particles in the wake (Figure taken from [22])

doublet distribution across the panel would affect the strengths of the vortex particles obtained at the centre of trailing edge segments after conversion.

Method IV implements a well structured algorithm of converting the panels into vortex particles instead of directly utilizing the panel vortices into vortex particles in the wake. By ensuring the conversion to be dependent upon the length of the edges of the panels, every panel on the blade can have its own distinct strength of vorticity associated with it before being converted into vortex particles in the wake.

It is apparent from the above methods that once the particles are released into the wake they can be allowed to evolve to simulate the wake flow. This is the idea that is focused on in this thesis. However, the blade of the wind turbine has been chosen to be modelled with an actuator line of particles that will be shed into the wake. Therefore, the ideas presented in the methods presented above will serve to be helpful in understanding the steps involved in determining the strength of the vortex particles that will be shed from the actuator line.

3

The Vortex Particle Method (VPM)

We know that an incompressible fluid's motion can be described by the Navier-Stokes equations as given by equation 3.1-3.2. Taking the curl of this equation results in what is known as the velocity-vorticity equation as represented in equation 3.3, the derivation of which is presented in Appendix 1. This equation describes the flow field in terms of the rate of change of vorticity in the flow. It should also be noted that the equation is lagrangian in nature.

$$\nabla \cdot \vec{u} = 0 \tag{3.1}$$

$$\frac{\partial u}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = g - \frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{u} \tag{3.2}$$

$$\frac{D\vec{\omega}}{dt} = (\vec{\omega} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{\omega} \tag{3.3}$$

On interpreting the equation we see that on the left hand side we have the material derivative of the vorticity, $\vec{\omega}$ and on the right hand side we have the sum of two terms, the vortex stretching term and the diffusion term. The vortex stretching term is what is represented by $(\vec{\omega} \cdot \nabla)u$. Vortex stretching refers to the behaviour of particles that lead to the changes in vorticity due to their expansion and contraction in space. This term results in the amplification of vorticity when $\vec{u}$ and $\vec{\omega}$ are aligned parallely. In the diffusion term, ν is the kinematic viscosity and ∇^2 is the laplacian operator.

3.1. 2D and 3D vortex methods

In equation 3.3 it is to be noted that for a 2D flow (with z representing the third dimension), $\vec{\omega}_x = 0$, $\vec{\omega}_y = 0$ and $\frac{\partial}{\partial z} = 0$. This means that the vortex stretching term becomes :

$$(\vec{\omega} \cdot \nabla) = (\omega_x \frac{\partial}{\partial x} + \omega_y \frac{\partial}{\partial y} + \omega_z \frac{\partial}{\partial z})\vec{u} = 0 \tag{3.4}$$

So, the velocity-vorticity equation now becomes

$$\frac{D\vec{\omega}}{dt} = \nu \nabla^2 \vec{\omega} \tag{3.5}$$

Table 3.1 summarizes the velocity-vorticity equation for 2D and 3D viscous and inviscid flows.

	Viscous flow	Inviscid flow
2D	$\frac{D\vec{\omega}}{dt} = \nu \nabla^2 \vec{\omega}$	$\frac{D\vec{\omega}}{dt} = 0$
3D	$\frac{D\vec{\omega}}{dt} = (\vec{\omega} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{\omega}$	$\frac{D\vec{\omega}}{dt} = (\vec{\omega} \cdot \nabla)\vec{u}$

Table 3.1: Velocity-vorticity equation for 2D and 3D flows

3.2. The Biot-Savart law

We know that the Navier-Stokes equations are written in velocity terms and we also note the the velocity-vorticity form of the equation describes the fluid flow in terms of vorticity. This does not remove the requirement to determine the velocity when we have the Navier-Stokes equation in the velocity-vorticity form. The relation between velocity and the vorticity is given by the Biot-Savart law, equation 3.6.

$$d\vec{u} = \frac{\Gamma}{4\pi}\frac{d\vec{l}\times\vec{r}}{|\vec{r}^3|} \tag{3.6}$$

In the above equation, if Γ represented the circulation of a vortex element and if $d\vec{l}$ was the segment of the vortex element, then the velocity induced by the circulation at a position vector $\vec{r}$ is given by equation 3.6. This equation is presented here to help in understanding the influence of the Biot-Savart kernel that appears in the following sections.

3.3. Singular vortex methods

For incompressible flow, the vorticity is computed as the curl of velocity. Here, the velocity is an unknown quantity. Therefore, the velocity is computed as the curl of the streamfunction that satisfies $\nabla^2\psi = -\vec{\omega}$, where, $\vec{\omega}$ is obtained from:

$$\vec{\omega} = \sum_p \vec{\omega}^p \delta(\vec{x}-\vec{x}^p) vol^p = \sum_p \alpha^p \delta(\vec{x}-\vec{x}^p) \tag{3.7}$$

According to Winckelmans [35],

$$\psi = G(\vec{x}) * \vec{\omega} = G(\vec{x}-\vec{x}^p) * \alpha^p = \frac{1}{4\pi}\sum_p \frac{\alpha^p}{|\vec{x}-\vec{x}^p|} \tag{3.8}$$

where, $G(\vec{x}) = \frac{1}{4\pi(|\vec{x}|)}$ is the green's function for $-\nabla^2$ and $*$represents convolution.
By computing the velocity as the curl of the streamfunction, ψ, we would obatain:

$$\vec{u} = \nabla\times\psi = \sum_p \nabla G(\vec{x}-\vec{x}^p)\times\alpha^p \tag{3.9}$$

$$\vec{u} = -\frac{1}{4\pi}\sum_p \frac{\vec{x}-\vec{x}^p}{|\vec{x}-\vec{x}^p|^3}\times\alpha^p = \sum_p K\times\alpha^p \tag{3.10}$$

In equation 3.10 K$\times$ represents the Biot-Savart kernel.

After calculating the velocity using kernel K and α^p, the vorticity of the particles are updated using equation 3.3. The velocity induced by a particle on every other particle in the domain of computation is singular at its position and its influence decays as the distance from the particle under consideration increases.

A major drawback of the vortex particle methods is that the vorticity field is not divergence free at all times as,

$$\nabla\cdot\vec{\omega} = -\frac{1}{4\pi}\sum_p \frac{\vec{x}-\vec{x}^p}{|\vec{x}-\vec{x}^p|^3}\cdot\alpha^p \tag{3.11}$$

However, the vorticity field can be made divergence free by rewriting it as in equation 3.12:

$$\vec{\omega}^p = \sum_p [\alpha^p\delta(\vec{x}-x^p) + \nabla(\alpha^p\cdot\nabla\frac{1}{4\pi|\vec{x}-\vec{x}^p|})] \tag{3.12}$$

After expansion of the gradient terms in equation 3.12, Winckelmans [35] rewrites this as:

$$\vec{\omega}^p = \sum_p [(\delta(\vec{x}-x^p) - \frac{1}{4\pi|\vec{x}-\vec{x}^p|^3})\alpha^p + 3\frac{((\vec{x}-\vec{x}^p))\cdot\alpha^p)}{4\pi|\vec{x}-\vec{x}^p|^5}(\vec{x}-\vec{x}^p)]] \tag{3.13}$$

3.3.1. 3D Biot-Savart kernel

For three dimensional flows each component of the induced velocity would be due to two components of vorticity of the particles. Hence the Biot-Savart kernel $K\times$ can be noted to become:

$$K = \frac{1}{4\pi|\vec{x}|^3}\begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix}$$

3.4. Regularization of velocity-vorticity equation

The singular VPM provides the general idea behind the VPM. As stated before, this formulation generates a singularity at the particle and a decaying influence as the distance from the particle increases. To avoid the singularity, we regularize the kernels in such a way that it produces zero influence at the particle instead of a singularity. For this purpose we define a regularization function ζ_σ with σ denoting the smoothing radius or the blob size, such that:

$$\omega_\sigma = \zeta_\sigma * \omega = \sum_p \alpha^p \zeta_\sigma(\vec{x} - \vec{x}^p) \tag{3.15}$$

Where, $\zeta_\sigma = \frac{1}{\sigma^3}\zeta(\frac{|\vec{x}|}{\sigma})$

As in the case of singular formulation the velocity is obtained as a curl of ψ which satisfies $\nabla^2\psi = -\omega$. The velocity field is now obtained as:

$$\vec{u}_\sigma = \nabla \times \psi_\sigma = \sum_p K_\sigma \times \alpha^p \tag{3.16}$$

with $K_\sigma = -\frac{q_\sigma(\vec{x})}{|\vec{x}^3|})$, where, $q_\sigma = q(\rho) = \int_0^\rho \zeta(t).t^2 dt$

3.5. Discretizing the viscous diffusion term

The idea behind discretizing the diffusion term was to achieve it by redistributing particle strengths α^p among the particles [35], where the superscript p represents the index of particles. The laplacian in the diffusion term has been approximated using an integral operator which is then discretized using the particle representation.

The Laplacian of a function $f(\vec{x})$ is approximated to be

$$\nabla^2 f(\vec{x}) \approx 2\int (f(\vec{x}^p) - f(\vec{x}^q))\eta_p(\vec{x}^q - \vec{x}^p)d\vec{x}^q \tag{3.17}$$

It has been shown by Winckelmans that a convection-diffsion equation of the form 3.18 can now be approximated with the integral as 3.19

$$\frac{\partial f}{\partial t} + \nabla\cdot(f\vec{u}) = \nabla^2 f \tag{3.18}$$

$$\frac{\partial f}{\partial t} + \nabla\cdot(f\vec{u}) = 2\int (f(\vec{x}^p) - f(\vec{x}^q))\eta_p(\vec{x}^q - \vec{x}^p)d\vec{x}^q \tag{3.19}$$

With particle approximation 3.17 can be represented as 3.20. Winckelmans has shown that using equation 3.20, the convection-diffusion equation has been solved to produce the solution to the equation 3.19 as in 3.21-3.22.

$$f_\sigma = \sum_p (f^p vol^p)\zeta_\sigma(\vec{x}^q - \vec{x}^p) \tag{3.20}$$

$$\frac{d\vec{x}}{dt} = \vec{u} \tag{3.21}$$

$$\frac{d(f^p \mathrm{vol}^p)}{dt} = \frac{2\nu}{\sigma} vol^p \sum_p vol(f^q - f^p)\times\eta_\sigma(\vec{x}^p - \vec{x}^q) \tag{3.22}$$

Using this method, the scheme for the velocity-vorticity equation becomes:

$$\frac{d\vec{x}}{dt} = \vec{u} \tag{3.23}$$

$$\frac{d\alpha^p}{dt} = (\alpha^p \cdot \nabla)\vec{u} + \frac{2\nu}{\sigma} vol^p \sum_p vol(\omega^q - \omega^p) \times \eta_\sigma(\vec{x}^p - \vec{x}^q) \tag{3.24}$$

3.6. Final discretized formulation

Winckelmans' high order regularization has the following definitions:

$$\zeta(\rho) = \frac{7.5}{4\pi(\rho^2+1)^{3.5}} \tag{3.25}$$

$$G(\rho) = \frac{\rho^2 + 1.5}{(\rho^2+1)^{1.5}} \tag{3.26}$$

$$q(\rho) = \frac{\rho^3}{4\pi(\rho^2+1)} \tag{3.27}$$

The high order regularization functions are benefial in the sense that they have the same convergence properties as that of a Gaussian regularization function. Applying the high-order regularization definitions gives us the descretized forms of the evolution equations as represented by 3.28 - 3.29.

$$\frac{d\vec{x}^p}{dt} = -\frac{1}{4\pi}\sum_q \frac{|\vec{x}^p - \vec{x}^p|^2 + 2.5\sigma^2}{(|\vec{x}^p - \vec{x}^p|^2 + \sigma^2)^{2.5}}(\vec{x}^p - \vec{x}^q) \times \alpha^q \tag{3.28}$$

$$\begin{aligned} \frac{d\alpha^p}{dt} = \frac{1}{4\pi}\sum_q \frac{|\vec{x}^p - \vec{x}^p|^2 + 2.5\sigma^2}{(|\vec{x}^p - \vec{x}^p|^2 + \sigma^2)^{2.5}}(\alpha^p \times \alpha^q) + \\ 3\frac{|\vec{x}^p - \vec{x}^p|^2 + 3.5\sigma^2}{(|\vec{x}^p - \vec{x}^p|^2 + \sigma^2)^{2.5}}(\alpha^p \cdot ((\vec{x}^p - \vec{x}^q \times \alpha^q))(\vec{x}^p - \vec{x}^q) + \\ 105\nu\frac{\sigma^4}{(|\vec{x}^p - \vec{x}^q|^2) + \sigma^2)^{4.5}}(vol^p\alpha^q - vol^q\alpha^p) \end{aligned} \tag{3.29}$$

3.7. Algorithm for VPM

Figure 3.1 provides a simple flowchart for the VPM algorithm.

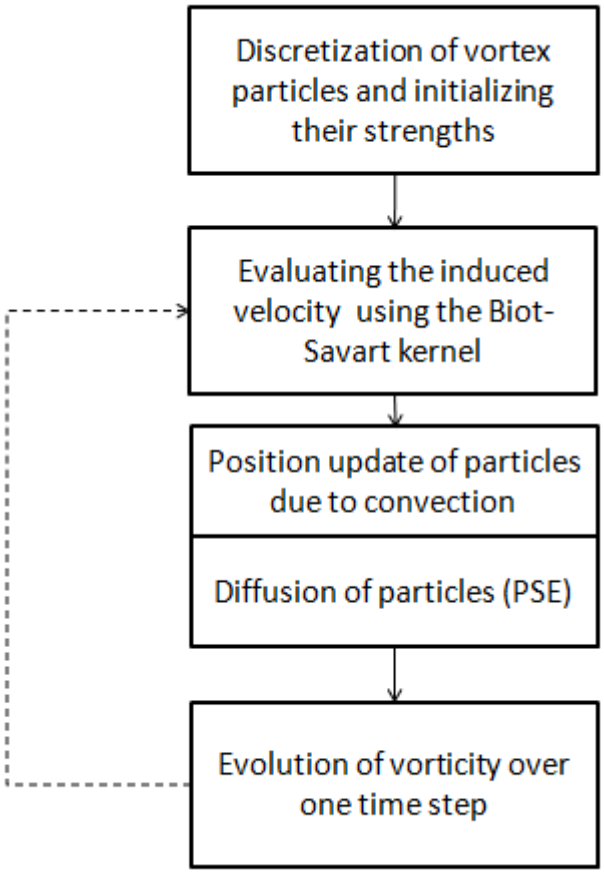


Figure 3.1: Algorithm for VPM

4

The Fast Multipole Method and the BBFMM3D library

4.1. Essential overview of the Fast Multipole Method (FMM)

This section gives a brief overview of the fast multipole method needed to understand it to the extent required for this thesis. For a detailed understanding of the method [14] can be referred. In general when we have an n-body problem that necessitates the computation of a matrix-vector product, Kq, where K is a square matrix and q is a column vector, the operation is of the complexity $O(n^2)$. This can be noted from equation 4.1.

$$S = \sum_{j=1}^{n} K_{ij} q_j \text{ for } i = 1, ..., n \tag{4.1}$$

This is obviously a time consuming operation to be performed when n is very large, say, of the order of 10^5. The fast multipole method reduces this order of operation to $O(n)$. To illustrate the essential idea behind FMM, let us consider two domains A and B, as shown in figure 4.1, of clusters of particles with particles in A inducing velocity on each particle in B. Let us also further consider that there are n_A and n_B number of particles in A and B respectively. The complexity of the operation of inducing velocity in B by the particles in A is $O(n_A \times n_B)$. To bring down this level of complexity in algorithmic computation, FMM approximates the potentials of the particles in A to a single point, say, p_A by a process known as multipole expansion if the particles are close to one another such that this approximation does not induce any significant errors in the result. The induced velocities in B are then computed due to this one particle at p_A. Similarly, the particles in B are approximated (assuming they are closely positioned) at a point, say, p_B to reduce the complexity even further and computing the induced velocity on one particle in B. This induced velocity is then distributed over the originally closely spaced particles in B by a process known as local expansion. The order of this entire operation now becomes $O(n_A + n_B)$.

When we have n particles in a domain of computation where one particle interacts with every other particle in the domain the induced velocity is computed as given in equation 4.1. However, to apply FMM the domain is subdivided into clusters of particles. This subdivision is done by using an octree data structure for 3D domains. For convenience this is illustrated using a quad-tree data structure (2D domains) in the following paragraphs and an analogous understanding can drawn for the case of a 3D domain. In the following paragraphs levels, near field and far field are illustrated which are essential for understanding the algorithm of FMM.

Let us consider a 2D domain containing n particles in it as shown in figure 4.2a. This undivided domain is assigned the level 0. Levels represent the number of times the domain has been subdivided based on the tree structure. Level 0 refers to the undivided domain and it contains one cell. In a quad-tree data structure each cell in a level gets subdivided into four cells as the index of the level increases, i.e. level 1 contains four cells and level 2 contains sixteen cells and so on, as shown in figure 4.2. Each cell in a level is called a child of the larger cell in its previous (or higher) level. The near field of a cell are all the cells directly in contact with the cell in question and the remaining are all called as the far field. For example, in figure 4.3 the cell marked **'X'** has the cells N1 to N8 in its near field and the cells F1 to F7 in its far field.

The total induced velocity computed at a point is equal to the sum of the contributions from the near and the far field as given in equation 4.2. Only the summation involving the far field is accelerated using FMM while the near field is computed using the direct method.

$$v = \sum_{i=1}^{N} a_i K_{ij} + \sum_{i=1}^{F} a_i K_{ij} \tag{4.2}$$

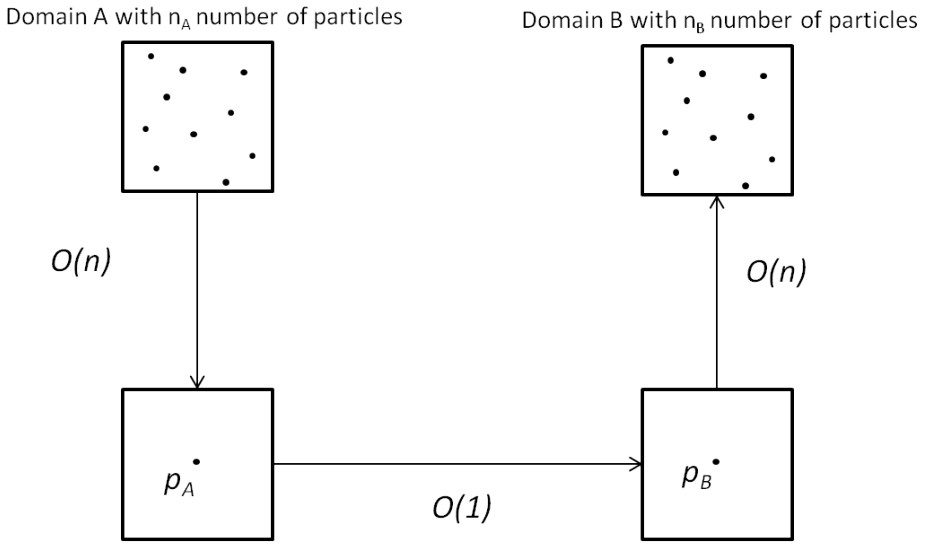
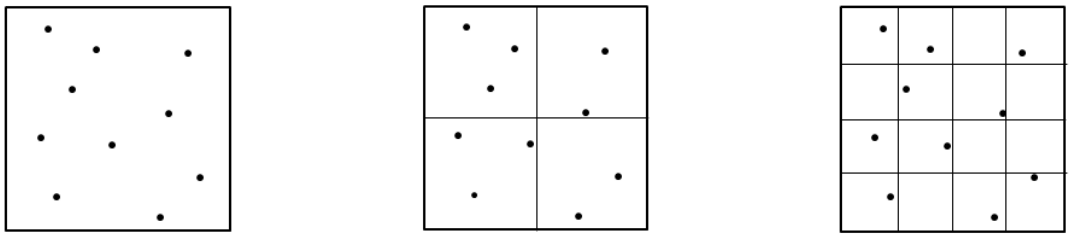


Figure 4.1: Basic idea behind FMM's reduction of algorithmic complexity to $O(n)$



(a) Level 0 (b) Level 1 (c) Level 2

Figure 4.2: Levels in the computational domain

4.2. Theory behind BBFMM (Black-Box FMM)

In the above section we saw that a matrix called the kernel matrix is needed to compute the matrix-vector product. This kernel matrix is generates element-by-element using the kernel functions. This can be quite time consuming. This is where the BBFMM comes in. This algorithm is kernel-independent. That is, the elements of the kernel matrix are interpolated using chebyshev polynomials with the interpolation being independent of what kernel is being used. Details on how the algorithm works has been presented in [12].

4.3. The BBFMM3D library

The BBFMM3D library, as mentioned before, implements the theory behind Black-Box Fast Multipole Method to approximate the matrix-vector product. A deeper explanation as to how the kernel matrix and column vectors are built will be explained in the following paragraphs. Then, the library is validated by calculating the same product in MATLAB. The accuracy and the time it takes to calculate the product using FMM and direct method are studied.

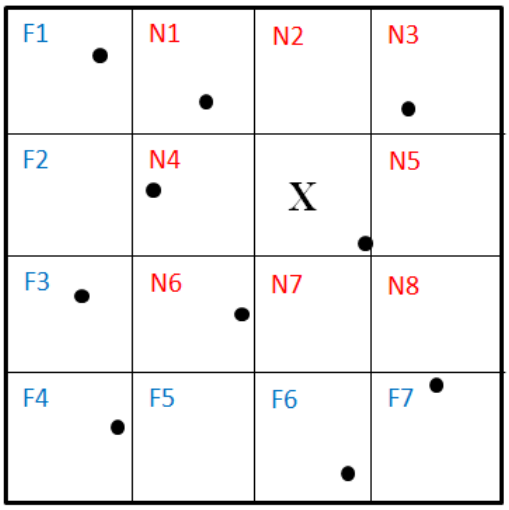


Figure 4.3: Near and far field at level 2

4.3.1. How the matrix and column vector are generated

In order to understand the working of the BBFMM3D library, one of the example programs, "binary_file _mykernel.cpp", was used. This example program generated the matrix-vector product, Kq, using randomly generated coordinates. The kernel function used for this purpose is defined as a class in the main program. Once the main() is executed and compute.hpp is run, the random coordinates are generated and stored in the respective variables. The elements of the Kernel matrix are generated element-by-element using these coordinates. The column vector, q, is also randomly generated. The number of points generated are according to what is defined in the metadata_test.txt file.

4.3.2. Validity of the results generated by BBFMM3D

As mentioned earlier, the main purpose behind implementing BBFMM3D to perform VPM computation is that it accelerates the matrix-vector multiplication. To make use of the BBFMM3D package to develop a VPM solver it was first necessary to validate the results, the matrix-vector product, that the package generated. The package comes with four sample programs that generate a matrix-vector product using a predefined (or user defined) kernel. The file binary_file_mykernel.cpp allows the user to define one's own kernel in the myKernel class defined in the file. However the example program already has the kernel $K = \frac{1}{r}$ defined in the myKernel class. This program was used to validate the code as it contained the user definable kernel class. Before running the code, the program was modified to make it write the coordinates of the random points it generates, the weights assigned to these points and the matrix-vector product into .txt files. The metadata_test.txt file was modified so that the program computed the matrix-vector product for 10 points to start with.

In order to validate the results obtained so far, a MATLAB code was written to compute the matrix-vector product using the direct method. This involved generating the matrix, element-by-element, using the $\frac{1}{r}$ kernel. $\frac{1}{r}$ was calculated for the coordinates obtained from the C++ code in .txt files. A column vector was generated by importing the .txt file containing the weights assigned to the coordinates. The result was computed using the matrix-vector product multiplication in MATLAB.

In order to validate the functioning of the library, the 'Relative Error' as computed by the program was compared with the relative error computed in MATLAB. The sample program computed the relative error, as in equation 4.3, by computing the sum of the square of the difference between the FMM results and direct results. This relative error was validated by computing it in MATLAB between the FMM results and the direct results obtained from MATLAB. The following plots and tables provide the matrix-vector product as generated by BBFMM3D (with FMM and Direct calculations) and MATLAB.

$$RelativeError = \sum_{i=1}^{N} \frac{(Kq_{i,fmm} - Kq_{i,dir})^2}{(Kq_{i,dir})^2} \tag{4.3}$$

Kq-FMM	Kq-DirectCalc	Kq-Matlab
7.94552	7.9456	7.94561
7.31632	7.31605	7.31605
8.98251	8.98007	8.98007
6.41785	6.41794	6.41793
8.94506	8.94455	8.94454
17.2953	17.2951	17.29511
9.70565	9.70679	9.70678
8.96205	8.96016	8.96015
9.75538	9.75528	9.75527
7.80905	7.80935	7.80934

Table 4.1: matrix-vector product for 10 particles

The relative error calculated by the sample program in BBFMM3D is as shown in figure 4.4.:

```
Reading pre-compute files ...
from compute.hpp-0.349749
Direct calculation starts from: 0 to 0.
Pre-computation time: 0.016
FMM computing time:   0
Exact computing time: 0
Relative Error: 0.000109232
```

Figure 4.4: Relative Error as computed by BBFMM3D for 10 particles

The relative error can be verified to be true by calculating it for the values tabulated in table 4.1 using equation 4.3. Now, the same error was computed between Kq-FMM and Kq-MATLAB. This relative error was computed to be 0.000109255329505864 which can be observed to be almost the same as computed by BBFMM3D.

To check for consistency, the relative error was computed for cases with 100, 500, 1000 and 5000 particles and the results are as tabulated in table 4.2. It can be observed that the relative error computed by BBFMM3D is consistent with those obtained in MATLAB and hence the functioning of the code has been validated.

No. of particles	Relative Error generated by BBFMM3D	Relative Error generated in MATLAB
10	10.92×10^{-05}	10.93×10^{-05}
100	3.85×10^{-05}	3.86×10^{-05}
500	2.58×10^{-05}	2.59×10^{-05}
1000	2.36×10^{-05}	2.36×10^{-05}
5000	2.08×10^{-05}	2.07×10^{-05}

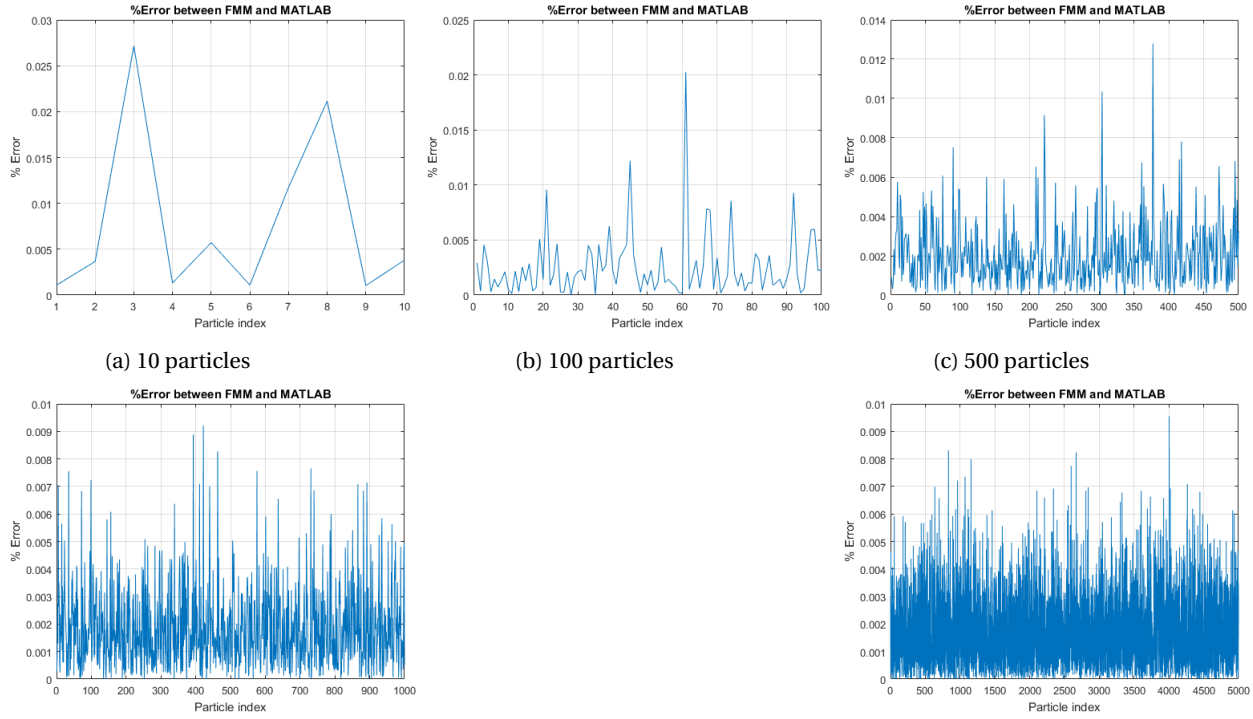
Table 4.2: Relative Errors

4.3.3. Accuracy of the results generated by BBFMM3D

The accuracy of the library was observed to be of the order of 10^{-4}. To measure the accuracy of the results the percentage error between the Kq-FMM and Kq-MATLAB results was computed. The following plots, in figure 4.5, reveals the maximum percentage error for every case tested, viz. with 10, 100, 500, 1000 and 5000 particles. It can be observed that the accuracy with the specified conditions is of the order of 10^{-4} for every case.

4.3.4. Features of BBFMM3D

This section concerns the variation of the FMM parameters and its consequences on the relative error and the time taken to complete the computation using FMM (and the exact method). The plots in figure 4.6c shows the variation of relative error with increasing number of particles in the domain of computation. It can be noted that the relative error decreases with increasing number of particles. This can be attributed to the fact that as FMM approximates the far field interactions, the approximations become more and more accurate as



(a) 10 particles

(b) 100 particles

(c) 500 particles

(d) 1000 particles

(e) 5000 particles

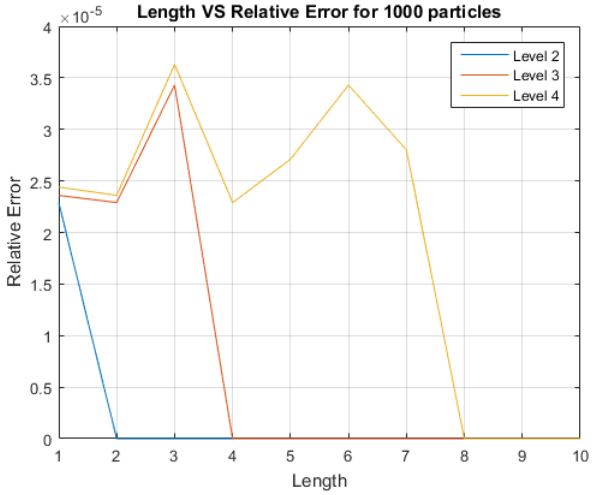
Figure 4.5: Percentage errors between FMM and MATLAB results for increasing number of particles for l=1 and level = 3

the number of particles closer to each increases as the total number of particles increase. In figure 4.6a we have the variation of the relative error between the FMM and exact results with respect to the length of the cube. It has to be noted here that the length of a cube is the cut off length that distinguishes the near field from the far field. This means as the length increases there will be fewer and fewer number of particles in the far field. We know that the purpose of FMM is to approximate only the far field. And therefore, if there is no far field, only the near field computation occurs which is computed with an SVD scheme. This drastically reduces the relative error and this is what is observed in the figure for length greater than 4 for level 3 and lengths greater than 8 for level 4. It has been observed that the length and the level are related as $\frac{length}{2^{level}} = 0.5$ to give an almost negligible relative error. This is presented in the table below. All lengths below the value specified in the table produce a significant order of error. This means that there are a significant number of particles in the far field whose interaction are being approximated with FMM for lengths lower than that value.

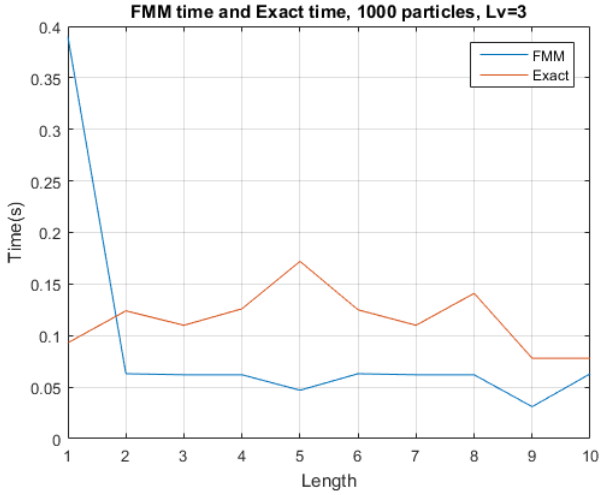
In figure 4.6b, we can see the time it takes to perform the operations using FMM and the exact method. This plot was obtained with 1000 particles in the domain with 3 levels. It can be observed that the time taken using the exact method is almost a constant for all values of lengths of the cube it was tested in. The time taken for FMM can be noted to be shorter than the time it took for the exact calculation and this is as expected of FMM due to its far field interaction approximation. To be able to see the difference between the time for FMM and exact calculation, figure 4.6d presents a clear distinction. This figure shows the time taken for the computation of the matrix-vector product that was performed for 15000 particles at level 3 in the domain. The time taken for exact computation almost remains a constant over all the length of the cell that it was tested upon. It can observed that at a cell length of 4, which is also where the relative error becomes negligible as seen in 4.6a, the FMM time almost remains a constant after increasing up to this length. Figure 4.6d clearly shows that the FMM library BBFMM3D indeed accelerates the evaluation of the matrix-vector product even when there are 15000 particles in the domain.

4.4. Conclusion

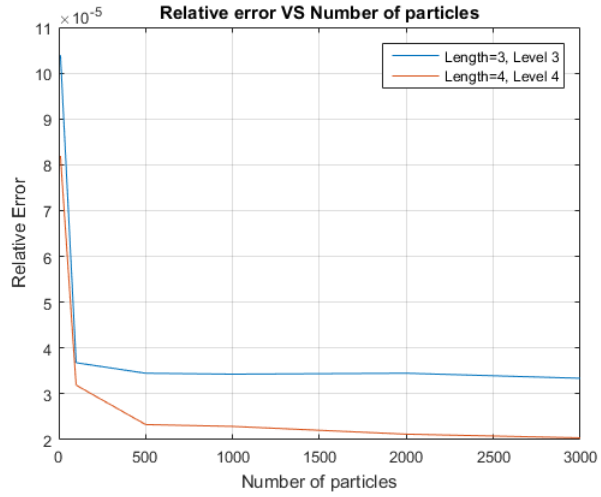
In this chapter the BBFMM3D library has been validated and its accuracy has been determined. It has also been observed that the cell's length that is being defined in the metadata.txt file defines the length of one



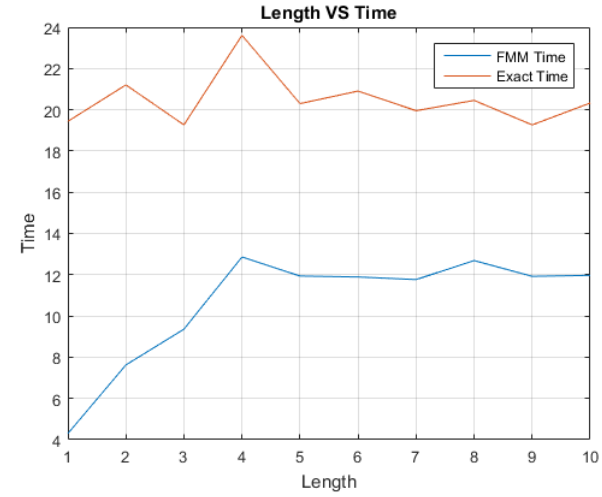
(a) Length VS Relative Error (for 1000 particles)



(b) Length VS Time (For 1000 particles, Level=3)



(c) Number of particles vs relative error



(d) Length VS time for 15000 particles, Level=3

Figure 4.6: Observations of BBFMM3D's features

cell that accounts for the cut off length to distinguish between the far field and the near field. There is also a relation between the level and length of a cell that defines the limit to the length of a cell for a particular level beyond which FMM does not perform its necessary function due to all points being in the near field. It has also been observed that the FMM library indeed accelerates the computation of the matrix-vector product. In the next chapter for the purpose of simulating the vortex rings we shall use the value of length = 2 for Level 3 as it has been observed to produce the lowest possible relative error when far field interactions are taken into account.

5

VPM Solvers

In order to solve the velocity-vorticity equation to accomplish the goal of our thesis, we need a VPM solver to begin with. This chapter concerns the development of such a solver that can solve the Navier Stokes equations for an unbounded flow.

Also, as explained in chapter 1, one of the goals of this project is to implement the FMM library to accelerate the computation of $O(n^2)$ calculations involved in VPM. The acquired FMM library (BBFMM3D) has been written in C++. In order to make the coding of VPM in C++ easier a MATLAB VPM solver, that solves the Navier-Stokes equation in Velocity-Vorticity form for the case of a single vortex ring and the cases of interaction of two vortex rings, has first been coded and the results were validated against the results from the theory of vortex rings. These results were used as a basis for comparison to validate the results obtained from using FMM.

The following sections of this document will present the details of the solver coded in MATLAB, the results and their validity, the results obtained from using the FMM library and how they compare with those obtained from MATLAB.

5.1. The MATLAB VPM solver

The MATLAB solver involved the definition of four functions to calculate the induced velocity, the vortex stretching term, the diffusion term and updating the circulation strength at the end of a time step. Although the velocity-vorticity equations look like they can be coded with ease in MATLAB, the formulation of the equations into a matrix-vector product was slightly challenging. Why do we want to formulate the equations in the form of a matrix vector product? Read on. It would help here to note that the matrix-vector product are of the form $K\alpha$, where K represents the kernel square matrix and α is the column vector representing the circulation strength of particles in the domain. Also, it is important to keep in mind that the results of the MATLAB code have been planned to be used as a basis for comparison to validate the BBFMM3D code. The BBFMM3D library can only be used to solve for a matrix-vector product where the matrix is a square matrix and the vector is a column vector. Therefore it was necessary to code the MATLAB solver in just the same way as BBFMM3D would solve the equations, by computing the matrix-vector product of terms involved in the equations, to be certain that there are no errors arising out of any mis-formulations between the codes written in the two languages.

5.1.1. Matrix-Vector product formulation

This section provides the matrix-vector product formulations of the evolution equations. For convenience the evolution equations for the vortex particle method are provided below again.

Induced velocity:

$$\frac{d\vec{x}^p}{dt} = \frac{-1}{4\pi}\sum_q \frac{|\vec{x}^p - \vec{x}^q|^2 + 2.5\sigma^2}{(|\vec{x}^p - \vec{x}^q|^2 + \sigma^2)^{2.5}}(\vec{x}^p - \vec{x}^q) \times \vec{\alpha}^q \tag{5.1}$$

Vorticity evolution:

$$\frac{d\vec{\alpha}^p}{dt} = (\vec{\alpha}\cdot\nabla)\vec{u} + \nu\nabla^2\vec{w} \tag{5.2}$$

where,

$$(\vec{\alpha}\cdot\nabla)\vec{u} = \frac{-1}{4\pi}\sum_q \frac{|\vec{x}^p-\vec{x}^q|^2+2.5\sigma^2}{(|\vec{x}^p-\vec{x}^q|\sigma^2)^{2.5}}(\vec{\alpha}^p\times\vec{\alpha}^q) + 3\frac{|\vec{x}^p-\vec{x}^q|^2+3.5\sigma^2}{(|\vec{x}^p-\vec{x}^q|^2+\sigma^2)^{3.5}}(\vec{\alpha}^p\cdot((\vec{x}^p-\vec{x}^q)\times\vec{\alpha}^q))(\vec{x}^p-\vec{x}^q) \tag{5.3}$$

and,

$$\nu\nabla^2\vec{w} = \frac{-1}{4\pi}\sum_q 105\nu\frac{\sigma^4}{(|\vec{x}^p-\vec{x}^q|^2+\sigma^2)^{4.5}}(vol^p\vec{\alpha}^q - vol^q\vec{\alpha}^p) \tag{5.4}$$

Using matrix-vector products the induced velocity is computed as:

$$U_x = (\frac{-1}{4\pi})((K\cdot\times dY)\alpha_Z - (K\cdot\times dZ)\alpha_Y) \tag{5.5}$$

$$U_y = (\frac{-1}{4\pi})((K\cdot\times dZ)\alpha_X - (K\cdot\times dX)\alpha_Z) \tag{5.6}$$

$$U_z = (\frac{-1}{4\pi})((K\cdot\times dX)\alpha_Y - (K\cdot\times dY)\alpha_X) \tag{5.7}$$

Here, K represents the kernel defined by:

$$K = \frac{(|\vec{x}^p-\vec{x}^q|^2+(\frac{5}{2})\sigma^2}{((|\vec{x}^p-\vec{x}^q|^2+\sigma^2)^{2.5}} \tag{5.8}$$

and the $\cdot\times$ operator defines an element-by-element multiplication with dX,dY or dZ which results in a square matrix. This square matrix is then matrix multiplied with α_X, α_Y and α_Z which are column vectors.

The vortex stretching and the diffusion terms can be similarly reduced to matrix-vector products.

$$VS_x = \alpha_y\cdot\times(K1\alpha_z) - \alpha_z\cdot\times(K1\alpha_y) + [(K2\cdot\times dX\cdot\times dY)\alpha_z - (K2\cdot\times dX\cdot\times dZ)\alpha_y]\alpha_x \tag{5.9}$$

$$VS_y = -\alpha_x\cdot\times(K1\alpha_z) - \alpha_z\cdot\times(K1\alpha_x) + [(K2\cdot\times dY\cdot\times dZ)\alpha_x - (K2\cdot\times dY\cdot\times dX)\alpha_z]\alpha_y \tag{5.10}$$

$$VS_y = \alpha_x\cdot\times(K1\alpha_y) - \alpha_y\cdot\times(K1\alpha_x) + [(K2\cdot\times dZ\cdot\times dX)\alpha_y - (K2\cdot\times dZ\cdot\times dY)\alpha_x]\alpha_z \tag{5.11}$$

Here, K1 and K2 are kernels defined by:

$$K1 = \frac{|\vec{x}^p-\vec{x}^q|^2+2.5\sigma^2}{(|\vec{x}^p-\vec{x}^q|\sigma^2)^{2.5}} \tag{5.12}$$

$$K2 = \frac{|\vec{x}^p-\vec{x}^q|^2+3.5\sigma^2}{(|\vec{x}^p-\vec{x}^q|^2+\sigma^2)^{3.5}} \tag{5.13}$$

For diffusion:

$$Diffusion_x = vol\cdot\times(K3\alpha_x) + \alpha_x\cdot\times(K3Vol) \tag{5.14}$$

$$Diffusion_y = vol\cdot\times(K3\alpha_y) + \alpha_y\cdot\times(K3Vol) \tag{5.15}$$

$$Diffusion_z = vol\cdot\times(K3\alpha_z) + \alpha_z\cdot\times(K3Vol) \tag{5.16}$$

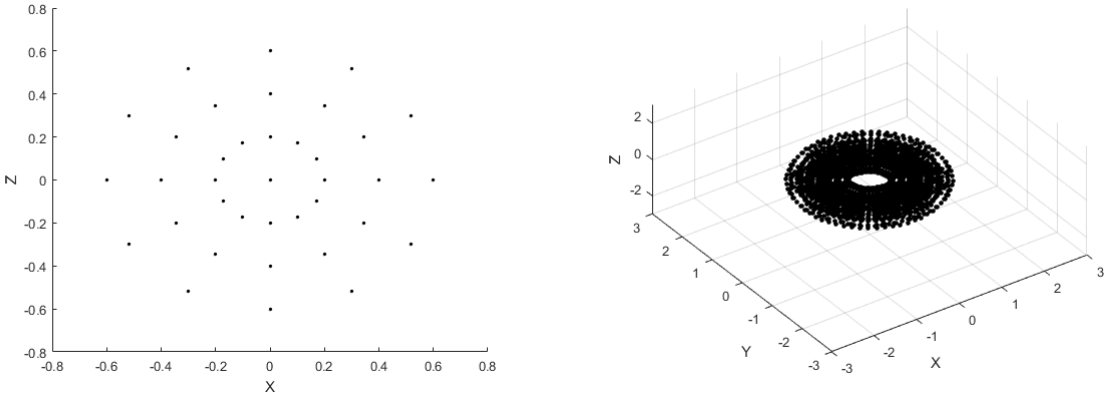
where K3 is:

$$K3 = 105\nu\frac{\sigma^4}{(|\vec{x}^p-\vec{x}^q|^2+\sigma^2)^{4.5}} \tag{5.17}$$

and *Vol* is the volume of the blob.

5.1.2. The domain of the vortex rings

Before we go into looking at the results of the test cases, here is a brief description of the domain of the vortex rings. The vortex ring is a toroidal structure and consists of vortex particles in the core. The core is generated with a line of particles placed between 0 and 0.6 with a spacing of 0.2. This line of particles is rotated about the Z axis to create the core as shown in figure 5.1a. This core is symmetrically distributed about the axis of the toroid as shown in Figure 5.1b. In the next section the initialization of the vorticity of particles in this domain has been discussed.



(a) A vortex ring core with 48 particles

(b) A vortex ring with 60 cores, 2880 particles

Figure 5.1: A vortex ring's core and a vortex ring

5.1.3. Initializing the vorticity of particles

The vorticity of the particles in the ring as shown in figure 5.1 has been initialized as explained in [35]. The distribution of vorticity is given by:

$$\vec{\omega}(\vec{x},0) = \frac{\Gamma}{2\pi\sigma^2}(1+\frac{r}{r}cos(\theta))e^{\frac{-r^2}{2\sigma^2}}\vec{e}_\phi \tag{5.18}$$

This is the distribution in one core in the azimuthal direction. Here R is the radius of the toroid, r is the distance of a particle in a core from the centre of the core, θ defines the angular position of a particle in the core, Γ is the circulation associated with the ring and σ is the blob size.
A point to note here is that due to memory constraints in MATLAB a vortex ring domain has been defined using only one particle at its core, thus simulating a very thin vortex ring. So the above mentioned vorticity distribution becomes:

$$\vec{\omega}(\vec{x},0) = \frac{\Gamma}{2\pi\sigma^2}\vec{e}_\phi \tag{5.19}$$

5.1.4. Validation case 1: Single vortex ring

A vortex ring with one particle at the core has been defined as shown below. The vorticity direction of the particles and the resulting direction of circulation of the vortex ring are shown in figure 5.3. The vortex ring was initialized with the following conditions: $\Gamma = 1$; $R = 1.0$; $\sigma = 0.1$; $R_e = 1e5$; $N = 180$;$dt = 0.1$. Running the VPM solver for this vortex ring with these conditions the evolution of the ring can be observed. Figures 5.4 show the evolution of the ring as it translates solely due to its induced velocity. It can be observed that the induced velocity in the vortex ring drops linearly over time as it undergoes diffusion and throughout its domain as expected. It can also be observed that instabilities develop throughout the ring before it undergoes diffusion completely.

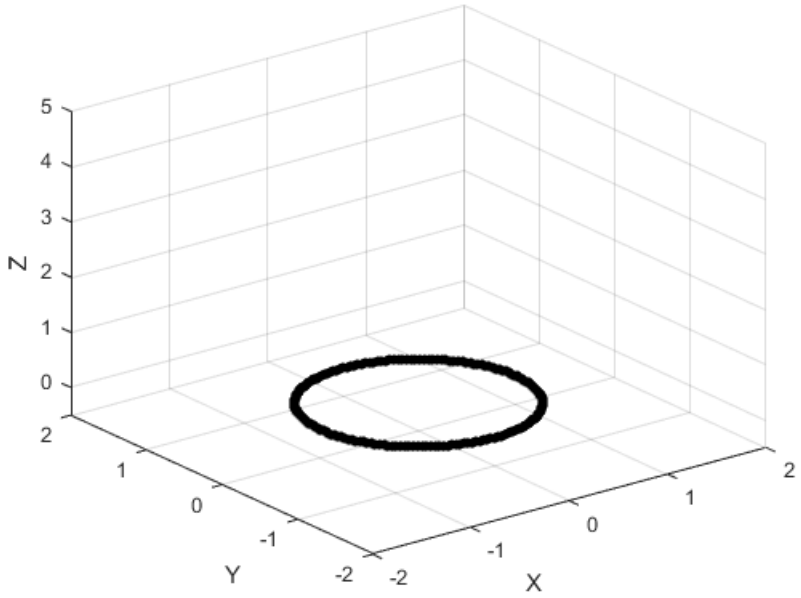


Figure 5.3: Vorticity and a vortex ring

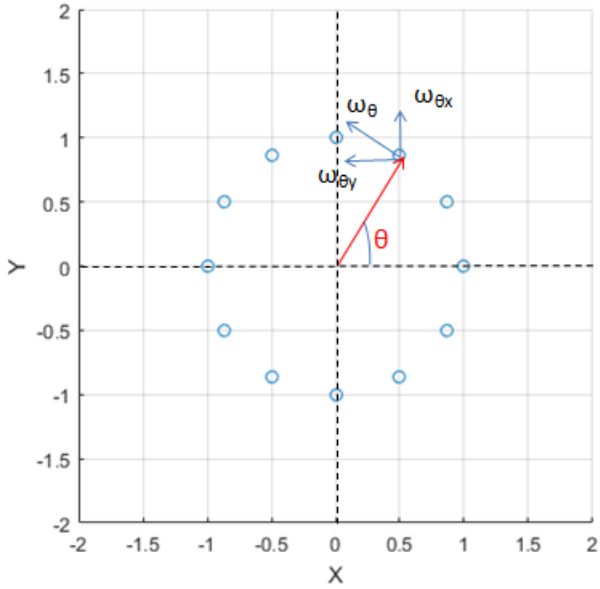
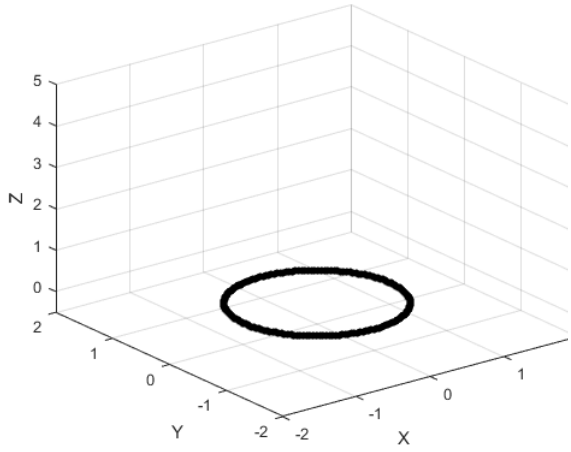
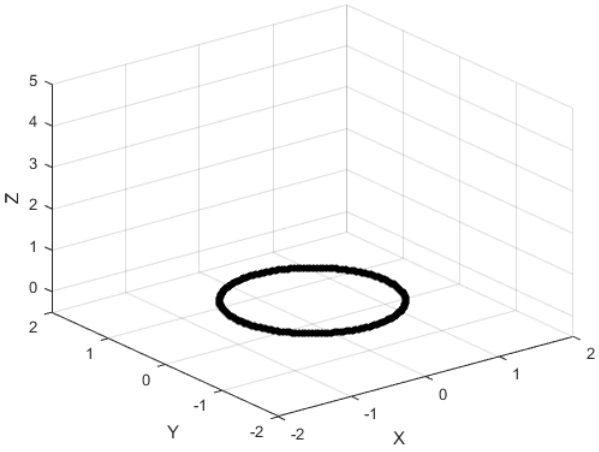


Figure 5.2: Vorticity direction of the particles in the ring

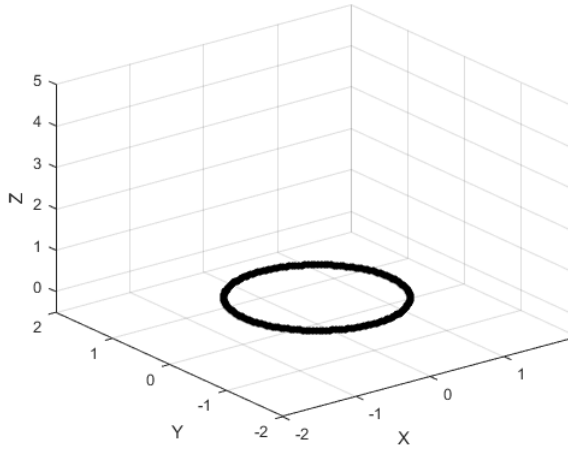
Pertaining to the validity of the VPM solver, to validate the VPM code the induced velocity of the ring that has been simulated in validation case 1 is compared with that of the ring simulated by Winckelmans [35]. Figure J.10 of [35] provides a plot for the variation of $\frac{4\pi R}{\Gamma} U_R$ versus the core size, σ_K. The initialization of the ring is done with $\Gamma = 1.0$, $R = 1.0$ with $N = 200$ computational points.The ring is simulated with identical initial parameters except with 180 computational points. The graph plotted by Winckelmans is presented in figure 5.5a, and represented as short dashes, while the result from the scheme implemented for the purpose of this thesis is presented in figure 5.5b.



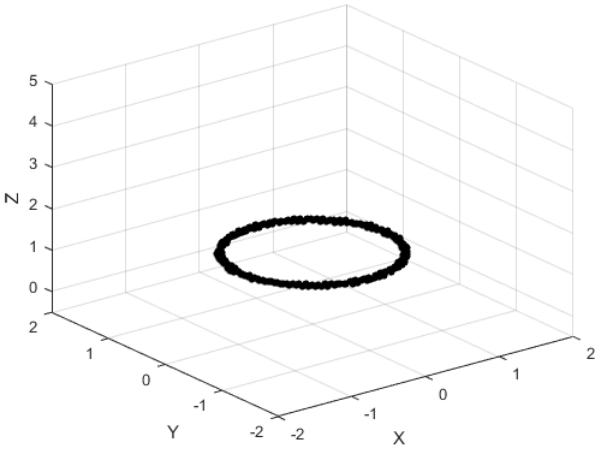
(a) at t=0



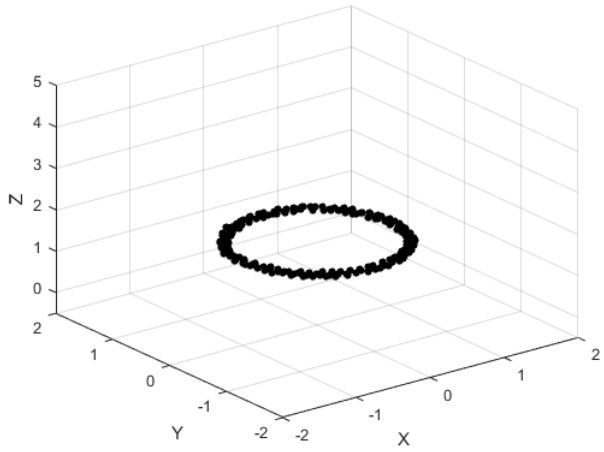
(b) at t=0.1



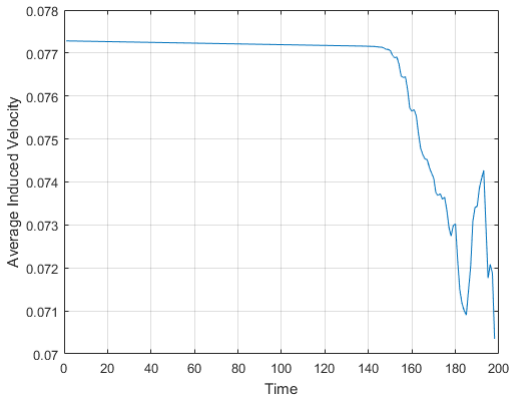
(c) at t=0.2



(d) at t=1.6

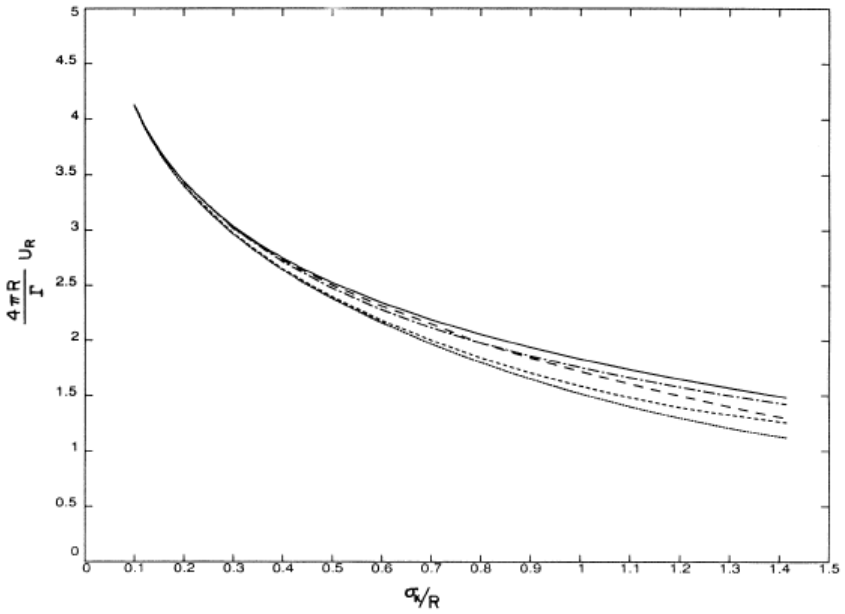


(e) at t=2

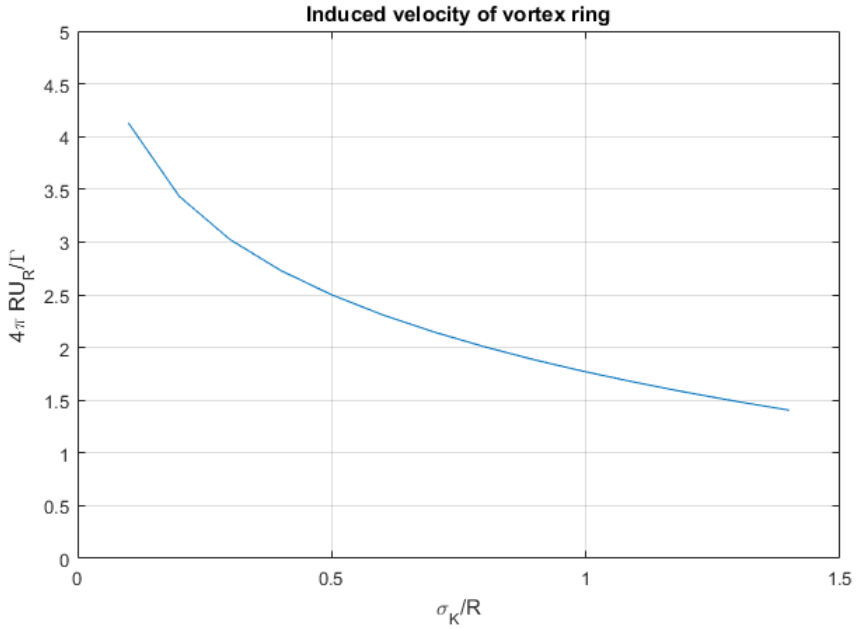


(f) Average Induced velocity of the ring

Figure 5.4: Translation of a vortex ring and its average induced velocity



(a) Variation of the induced velocity with the core size, from [35]



(b) Variation of the induced velocity with the core size

Figure 5.5: Induced velocity of the ring VS core size

It can be observed that, our results agree very closely with those that are presented in figure 5.5a. We can now say that the VPM code implemented in this thesis has been validated. In this regard, there are a few things to be noted. Firstly, the circulation of the ring Γ has been used to obtain the vorticity of the particles in the ring by computing it as $\frac{2\pi R\Gamma}{N}$. The cut-off radius for the regularization function is defined as $\sigma = \beta\sigma_K$, where, $\beta = e^{-0.75}$ and this is the β that is associated with the low-order algebraic kernel used in Winckelmans [35] simulation.

5.1.5. Validation case 2: Collision of two vortex rings

This validation case has been chosen to verify the validity of the diffusion term coded in the solver. For the case of the collision of two vortex rings we have two vortex rings positioned with their centers at 1.0 and 3.07 so that the minimum distance between the rings is 0.07 units. For the below mentioned conditions the results are as shown in below figures. Both rings have their vorticities aligned in the same direction. The vortex ring was initialized with the following conditions: $\Gamma = 1$; $R1 = R2 = 1.0$; $\sigma = 0.1$; $R_e = 10^5$; $N = 180 + 180$;$dt = 0.1$.

The correct behaviour of the diffusion term should allow the vortex rings to diffuse and merge into a single ring as the instabilities in the ring propagate from the points of contact of the two rings. An incorrect behaviour in the diffusion term would cause a numerical blow up of the simulation as instabilities in the rings develop.

The following figures (Figure 5.6 - Figure 5.7) show the evolution of the two colliding rings. As the two rings collide, they begin to merge at the point of contact between the two rings. It can be seen that the instabilities in the ring begin to propagate from this point towards the other end of the ring over time until they form a single ring. Also, the rings translate on account of their induced velocities while in the process of collision. The diffusion occuring during this process can also be clearly seen in the Figure 5.6 - Figure 5.7.

5.1.6. Validation case 3: Leapfrogging of two vortex rings

In this case two rings, with one having radius R1 =1.0 and the other R2 = 0.9 are placed concentrically. Both the rings have their vorticities aligned in the same direction. The vortex ring was initialized with the following conditions: $\Gamma = 1$; $R1 = 1.0$; $R2 = 0.9$; $\sigma = 0.1$; $R_e = 10^5$; $N = 180 + 180$;$dt = 0.1$. The following figures in Figure 5.9 show the evolution of leapfrogging of the two rings as they merge to form a single ring eventually.

It can be observed that the smaller ring has an induced velocity larger than that of the larger ring and quickly moves ahead of the larger ring while expanding. As it expands, the ring begins to slow down while the other ring contracts and speeds up to pass through the now larger ring. This process repeats until the two rings merge and diffuse. The rings expand and contract on account of the influence of the vortex stretching term causing an expansion and contraction of vorticity of the particles. The expansion and contraction can be visualized with the help of the cyclically varying diameter of the two rings as shown in Figure 5.8. The invariant quantities remain invariant as expected. This verifies the correct working of the vortex stretching term in the solver.

5.2. BBFMM3D solver

Once the MATLAB VPM solver was validated, the same scheme was implemented in the BBFMM3D code. For this purpose the case of the single vortex ring was simulated using BBFMM3D and was validated against the results obtained in MATLAB.

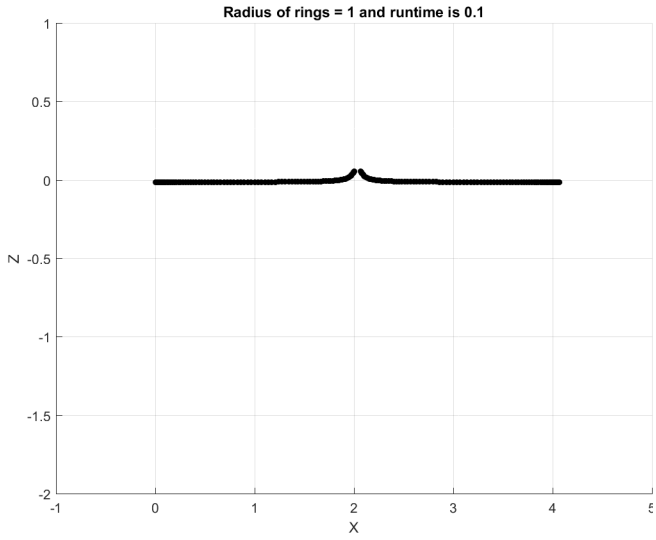
5.3. Error Analysis of the results from BBFMM3D

Now that the available BBFMM3D code has been validated, it is necessary to check for the validity of the results from the VPM solver coded using this library. To do this the error in the results of the terms involved in the evolution equations have been computed i.e. the error in induced velocity, error in the vortex stretching and diffusion terms. The following sections explain how this was done and present the results.

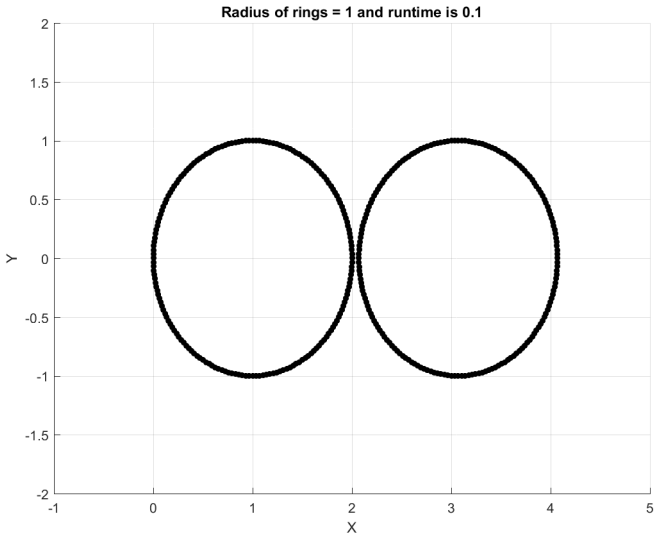
5.3.1. Error in the induced velocity for validation case 1

The error in the induced velocity is calculated by computing the percentage error between RMS (Root Mean Square) induced velocities from BBFMM3D and those from MATLAB. This error was computed over 4 time steps and it can be observed from figure 5.10 that the error converges over time.

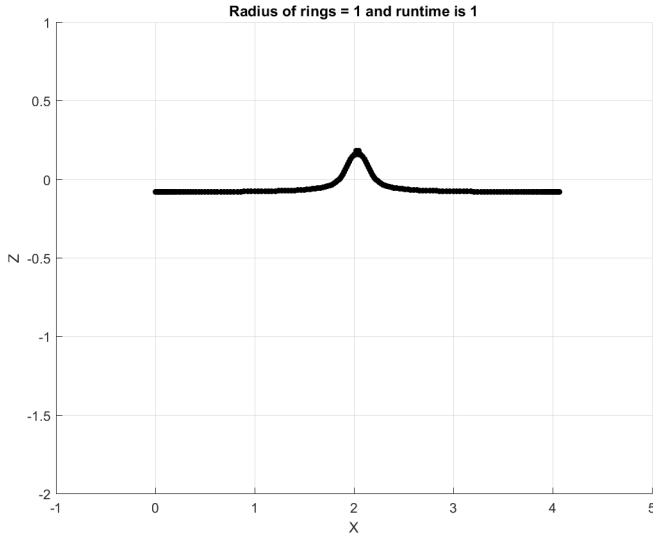
This suggests that the approximations that BBFMM3D makes to calculate the induced velocity over particles are acceptable and that the code written in BBFMM3D is yielding results as expected. This verifies the correct working of the BBFMM3D library to compute the induced velocity.



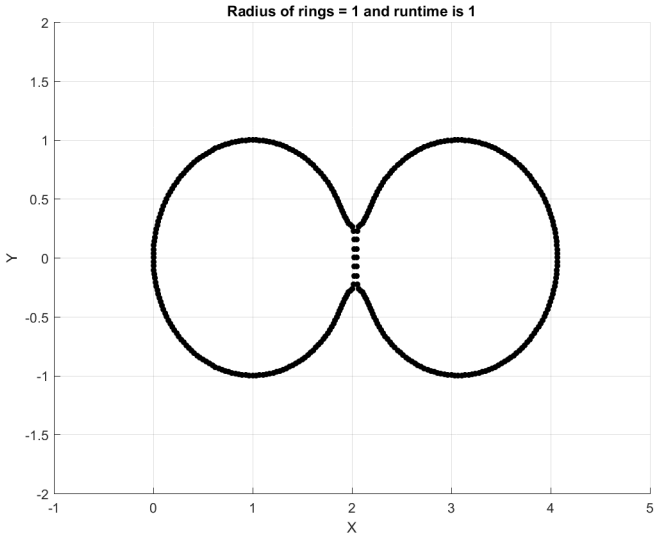
(a) at t=0.1



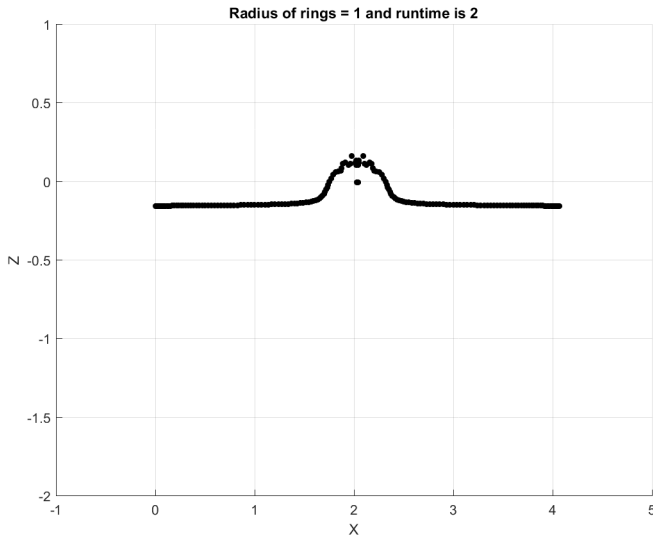
(b) at t=0.1



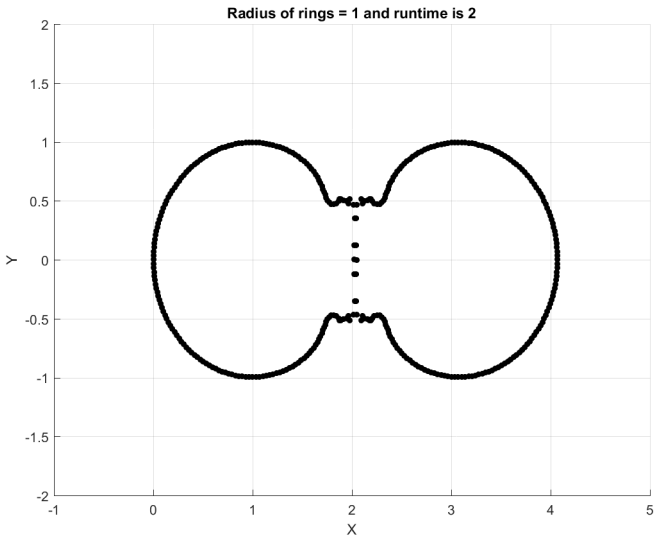
(c) at t=1



(d) at t=1

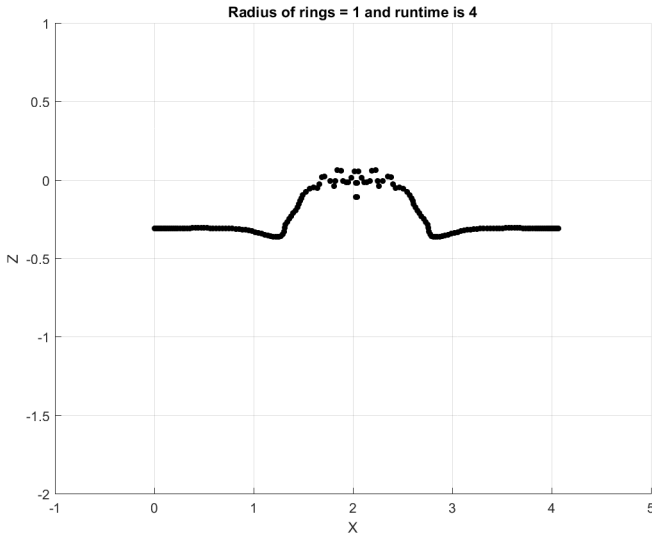


(e) at t=2

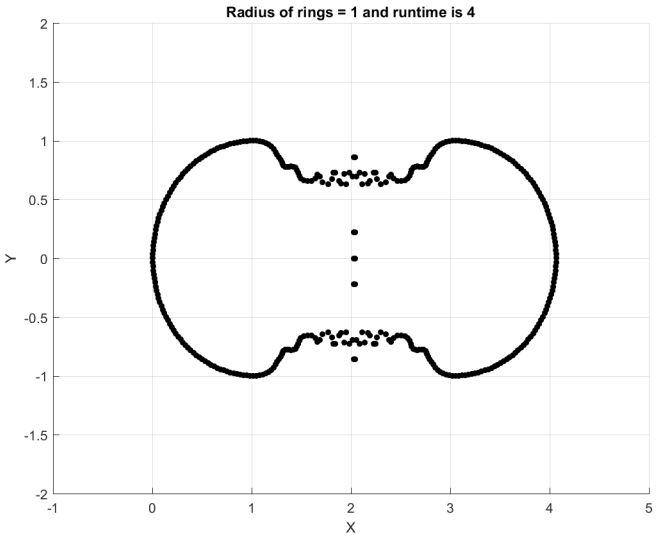


(f) at t=2

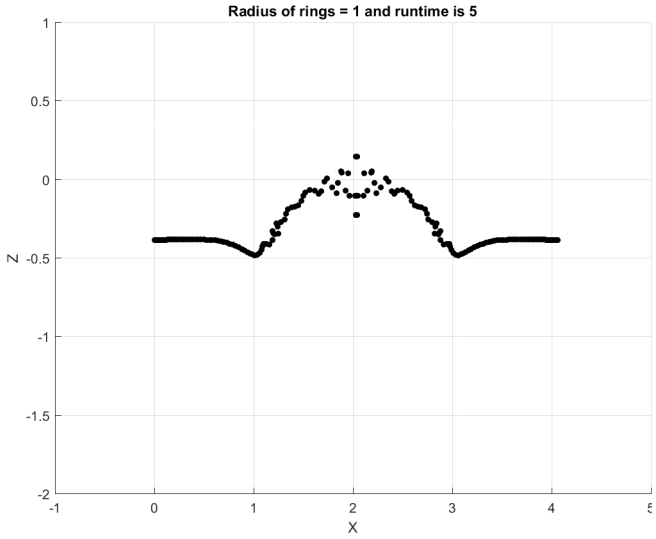
Figure 5.6: Collision of two vortex rings



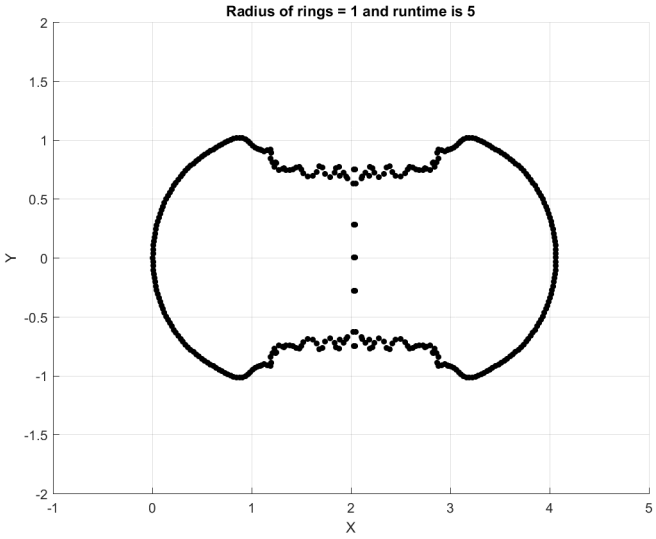
(a) at t=4



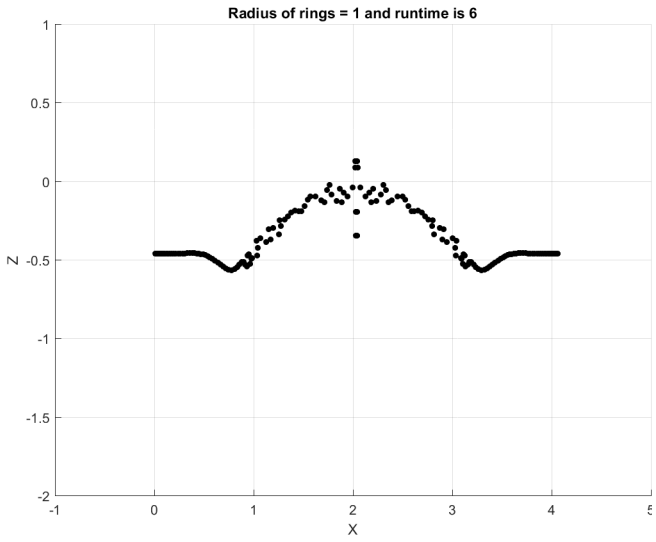
(b) at t=4



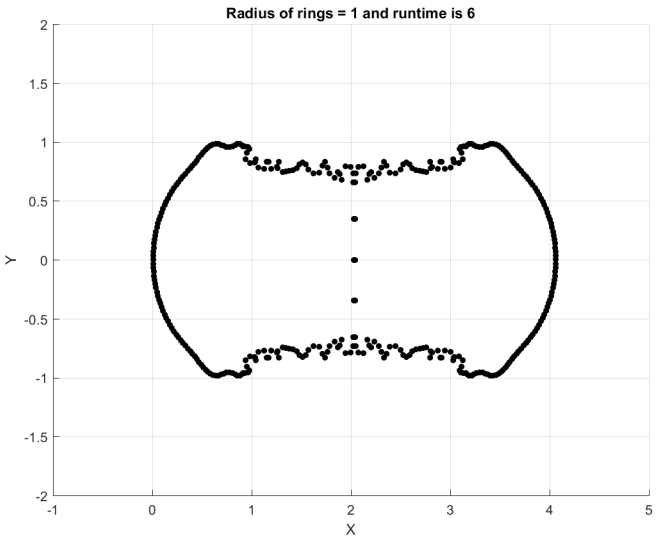
(c) at t=5



(d) at t=5



(e) at t=6



(f) at t=6

Figure 5.7: Collision of two vortex rings

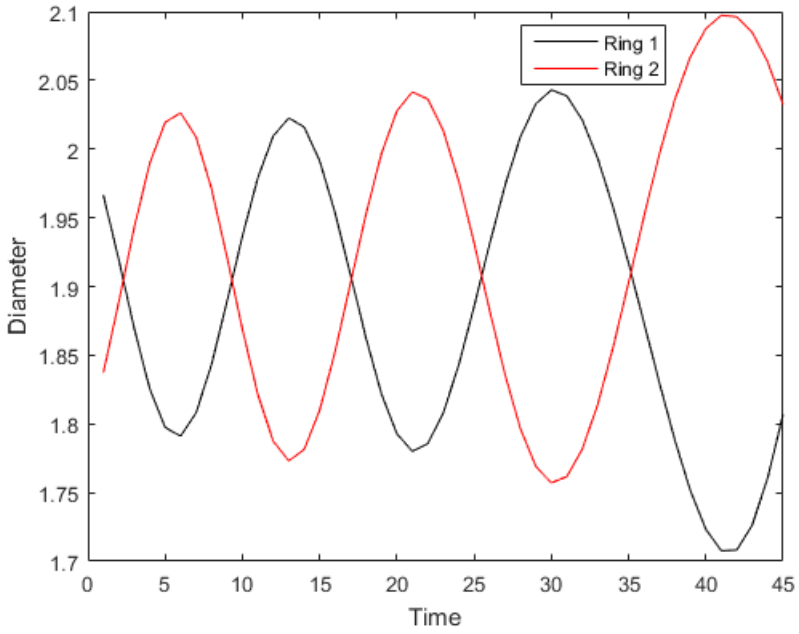


Figure 5.8: Variation of the diameter of the two interacting rings

5.3.2. Error in the vortex stretching term

The vortex stretching term in the equation 5.2 has been split into VS_A and VS_B terms with the former involving the products with K1 kernel and the latter with K2 kernel. Further VS_A has been split into VS_A1 and VS_A2 in order to be able to use the matrix-vector product formulation in BBFMM3D. The results computed by the BBFMM3D VPM solver for VS_A1 and VS_A2 are compared with those from MATLAB.

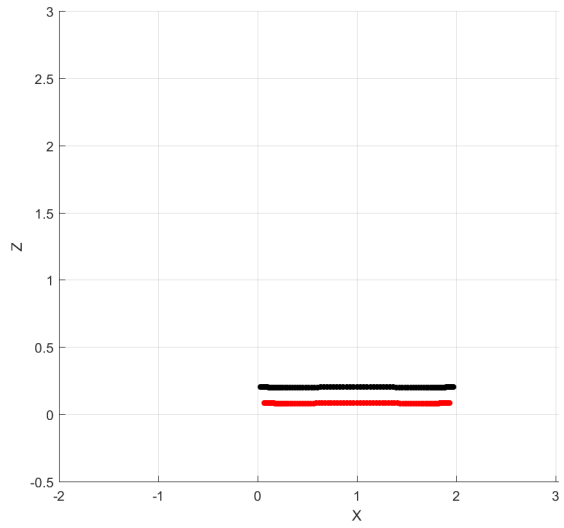
It can be observed that the product has an acceptable maximum error of 2%. However the term VS_A is computed as the difference between VS_A1 and VS_A2. It is important to note here that the latter terms only possess maximum of 7 decimals while the result for the same terms from MATLAB is accurate upto at least 15 decimals due to MATLAB's machine precision. Hence, the deference between VS_A1 and VS_A2, as far as, MATLAB is concerned produces a number of the order of 10^{-16} while BBFMM3D produces a number of the order of 10^{-7}. The result from BBFMM3D is relatively large when compared with the result from MATLAB. Due to the fact that MATLAB calculates upto machine precision and that C++ calculates only upto seven decimal places, while also approximating the results, it wouldn't be wise to compare the two results relatively. Doing so only gives us a large percentage error. However, observing that the terms VS_{A1} and VS_{A2} have a maximum percentage error of approximately 2% with respect to the results from MATLAB, we can remain confident with the fact that the computation of the vortex stretching is being done properly and that the relative error in it arises only from MATLAB's machine precision values. The validity of vortex stretching results can also be observed in the fact that the RMS induced velocity over increasing time converges to the MATLAB's computation of RMS induced velocity.

5.3.3. Conclusion

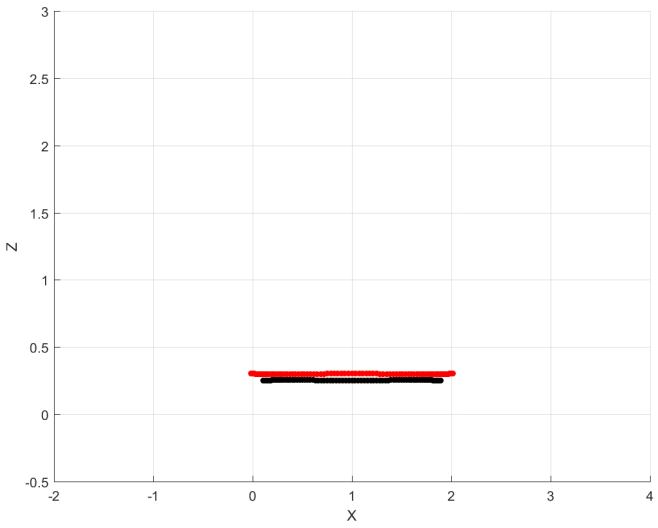
In this chapter we have seen the results of the MATLAB VPM solver exhibiting expected behaviour for the vortex rings which validates the MATLAB VPM Solver. Also, the error in BBFMM3D VPM solver's results, with respect to those of MATLAB, have been observed to be acceptably small in the case of induced velocity. However, for the case of vortex stretching the results cannot be compared relative to those from MATLAB as the values are approximated and truncated at seven decimals in the case of C++. But it can be agreed upon that as the individual terms in the equations yield almost the same results as MATLAB, the BBFMM3D code is suitable to simulate the fluid flow in the wake of a wind turbine for this project.

From the conclusions derived from the previous chapter with regards to BBFMM3D, we set the metadata

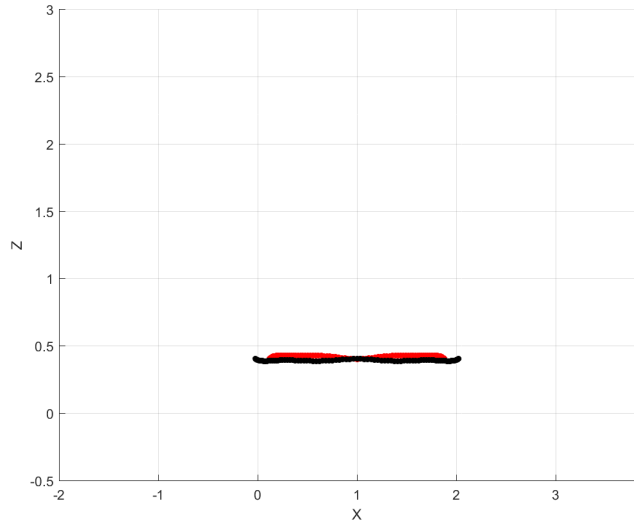
file to the corresponding numbers (Length=2, Level = 3, Chebyshev nodes = 4, Degree of freedom for source and field = 1) that generated the lowest relative error.



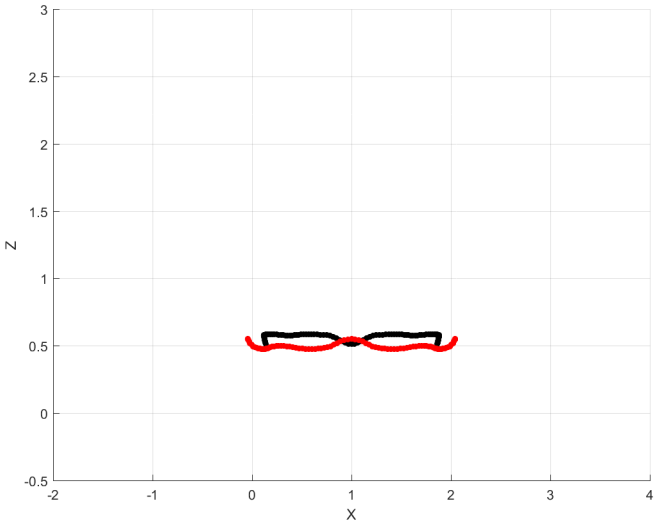
(a) at t=1



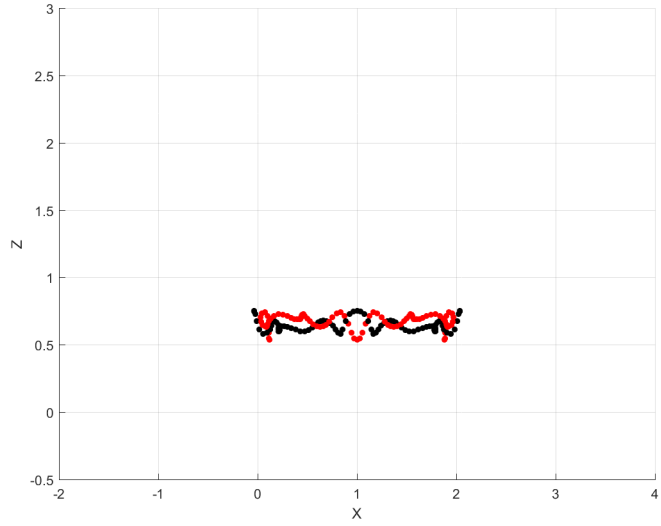
(b) at t=2



(c) at t=3



(d) at t=4



(e) at t=5

Figure 5.9: Leapfrogging of two vortex rings captured at intervals of 10 time-steps, dt=0.1

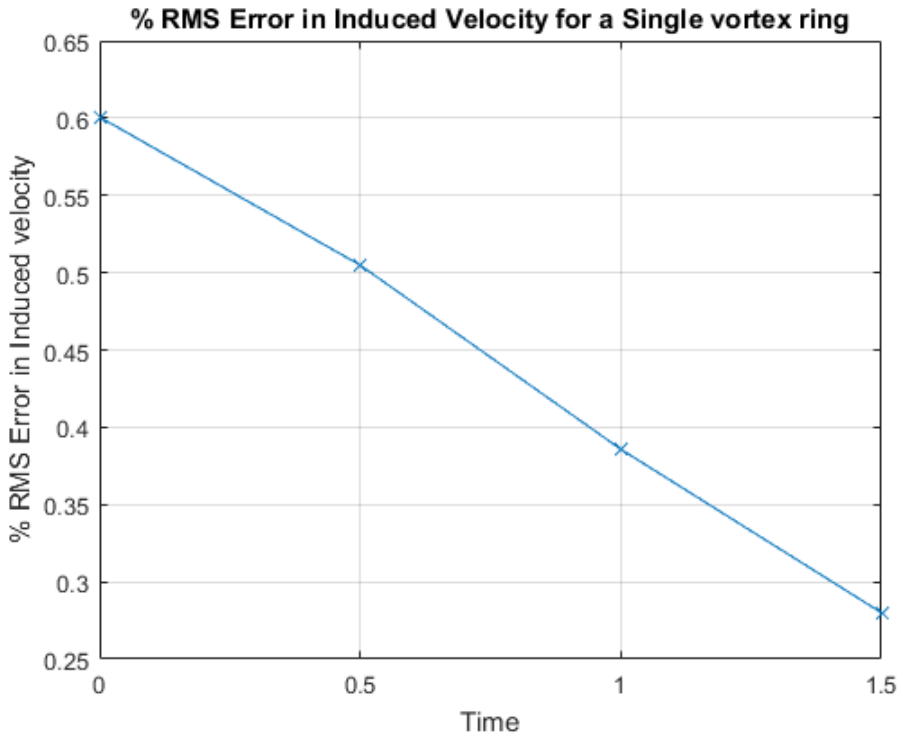


Figure 5.10: RMS induced velocity error

6

MATLAB wind turbine wake solver

Thus far the ingredients needed for the development of a wind turbine wake's solver using VPM have been set in place. We make use of the VPM segment developed in MATLAB to write a solver for the wake flow.

6.1. MATLAB VPM wind turbine wake solver

The idea behind the wind turbine wake's solver is the implementation of the actuator line method combined with an immersed lifting line and allowing the shed vortex particles from the actuator line to evolve using the VPM algorithm. The model of the turbine is obtained from [13]. From the data available on loading at discrete locations on the blade, the circulation at these locations were obtained from the Kutt-Jukowski theorem. In our solver we place the values of circulation at the same discrete locations (with respect to the centre of the rotor) and allow these points to be shed at every time step. During the same time step the discrete blade points are allowed to rotate through a prescribed angle. A point to note here is that as the time progresses more and more particles are shed into the domain. This reduces the computational speed and the solver almost down to a halt when there are more than 1000 particles in the domain.

6.1.1. Initializing the solver

Initializing the solver involved two parts: First part involved initializing the wind turbine's design parameters using the actuator line model along with the experimental conditions and the second part involved initializing the vorticity of particles that are shed in the wake.

Table 6.1 shows the experimental conditions at which the solver is initialized in comparison to the experimental conditions as in [13]. RPM and wind speed are rounded off for convenience.

The time step for every shedding of particles is calculated as $dt = 0.1666(\frac{\phi}{RPM})$, where ϕ is the angle of

	Radius (m)	RPM	Wind speed (m/s)	Tip-speed ratio (derived)
WT VPM solver	5.030	70	7	5.267
Conditions from [13]	5.030	72	6.7	5.417

Table 6.1: Input conditions for the solver

rotation of the blades in one time-step and this is set to 10°. Note that "1 shed" of particles means one set of vortex particles shed into the wake in one time-step (Corresponds to one row of red circles from figure 6.1).

The radius and windspeed were scaled to implement them in their non-dimensional forms while maintaining the same tip-speed ratio. The input paramenters then became as shown in table 6.2. Doing this proved to be erroneous as will be shown in the forthcoming sections in this chapter.

As explained in the above paragraph, to initialize the wind turbine solver, the initial values of the vortic-

	Radius	RPM	Wind speed	Tip-speed ratio (derived)
WT VPM solver	1	70	1.41	5.199
Conditions from [13]	5.030	72	6.7	5.417

Table 6.2: Non-dimensional input conditions for the solver

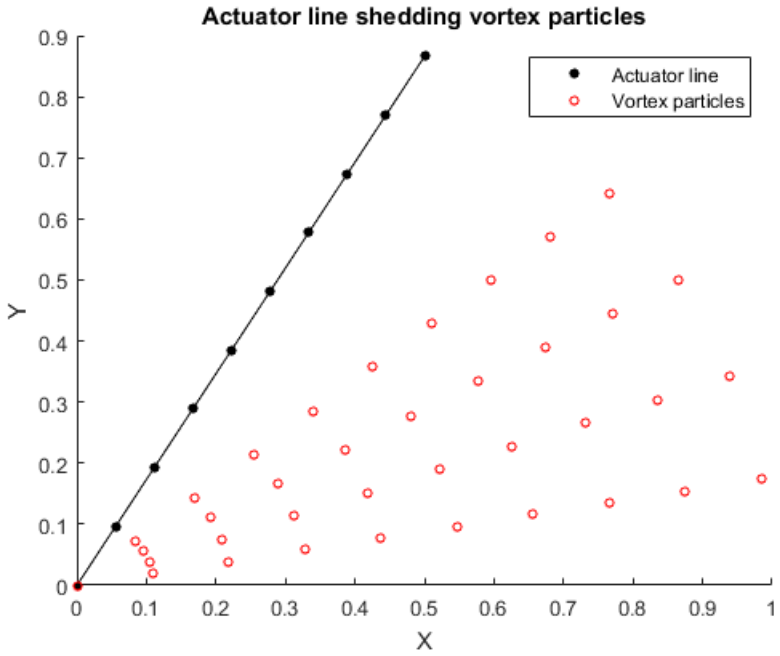


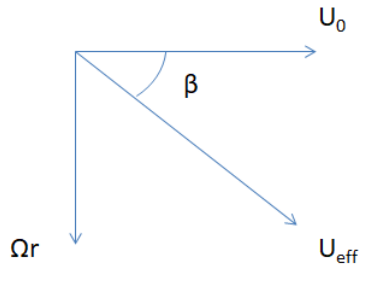
Figure 6.1: Actuator line with trailing vortex particles

ity in the wake are also required upon which the velocity-vorticity equation would be used to observe the evolution. The incoming air flow is irrotational and therefore does not have any vorticities. The vorticities in the wake are due to the vorticities shed by the Wind turbine blade. According to the lifting line theory for wings the wing is replaced by a line, at the quarter chord point, of point vortices and this is shed downstream in the form of trailing vortices as shown in figure 6.1. For this, we need the values Γ at discrete radial locations on the blade. We know that Kutta-Jukowski theorem relates lift and Γ as $\vec{L} = \rho\vec{U} \times \Gamma\vec{e}_b$, and therefore the discrete Γ distribution can be obtained if we have the lift distribution at discrete radial locations. To obtain this lift distribution we refer to [13]. From this distribution the circulation at discrete sections is determined as explained above. C_l at thise locations is also obtained from the lift. However, to evaluate the C_l the velocity of the incoming flow in required which is calculated as the effective velocity experienced by the blade at the locations where the circulations are defined. Angular velocity of the blade is taken into account in this process along with the velocity of the freestream to determine the local effective angle of attack of the blade. The magnitude of the trailing vortices according to lifting line theory is $\frac{d\Gamma}{dy}dy$.

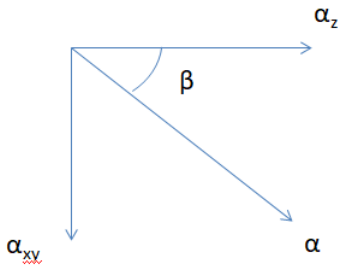
6.1.2. How the strengths for the vortex particles were implemented

To clearly understand how the strengths were implemented, see figure 6.2. In 6.2a, U_0 is the velocity of the incoming flow and Ωr is the angular velocity of the turbine and U_{eff} is the effective velocity of the wind as seen by the rotating turbine. Considering the trailing vorticities to be aligned parallelly with the effective wind direction, the trailing vorticities can be resolved into vectors α_z and α_{xy} as shown in 6.2b where α_z is the component of $\frac{d\Gamma}{dy}dy$ acting along the axis of the HAWT and α_{xy} includdes the components of $\frac{d\Gamma}{dy}dy$ acting in the rotational plane of the turbine. θ is the angle of rotation of the blade in one time-step and β represents in the direction of the trailing vorticity in the wake. Therefore, the components of vorticity in the wake become as written in equations 6.1-6.3.

$$\alpha_x = (\frac{d\Gamma}{dy}dy)\ \sin(\theta)\ \sin(\beta) \tag{6.1}$$



(a) Direction of effective velocity in the wake



(b) resolving the strength vector in the wake

Figure 6.2: Resolving the strength vector

$$\alpha_y = -(\frac{d\Gamma}{dy}dy)\ \cos(\theta)\ \sin(\beta) \tag{6.2}$$

$$\alpha_z = -(\frac{d\Gamma}{dy}dy)\ \cos(\beta) \tag{6.3}$$

It is also important to make this expression dependent on dt as the strengths of the particles depend on the frequency of the shedding taking place. For this purpose, we include a non-dimensionalized term that depends on dt into equations 6.1-6.3. The equations now become as in 6.4-6.6 (c is the mean chord length of the blade.

$$\alpha_x = (dt\ U_0/c)\ (\frac{d\Gamma}{dy}dy)\ \sin(\theta)\ \sin(\beta) \tag{6.4}$$

$$\alpha_y = -(dt\ U_0/c)(\frac{d\Gamma}{dy}dy)\ \cos(\theta)\ \sin(\beta) \tag{6.5}$$

$$\alpha_z = -(dt\ U_0/c)(\frac{d\Gamma}{dy}dy)\ \cos(\beta) \tag{6.6}$$

6.1.3. Discussion of the results pertaining to non-dimensionalized parameters

Figure 6.3 shows the wake obtained after non-dimensionalizing the parameters as in table 6.2. It can be observed that the wake expansion is not visible. Figure 6.4 shows the wake profile at 2D downstream of the windturbine. Here the diameter is considered as 2.0 due to non-dimensionalized radius. The plot clearly does not validate the code as the resulting plot is not what one would expect a wake to behave like. The wake must possess a velocity deficit on account of induction but clearly, the velocity has a surplus as we see the normalized velocity to be more than 1. Further discussion in section 6.1.4

6.1.4. Attempting to fix the possible errors

In an attempt to correct the results the idea of non-dimensionalization was discarded and the original values as in [13] were used. But doing resulted in a very large diameter (10 times as much) compared to the non-dimensional case and hence the wake had to have evolved up to at least 21m to study the profile at 2D. This corresponded to at least 150 sheds of particles as each shedding of particles would travel through 0.14m. As MATLAB performs the computation using the direct approach to solving the n-body problem involving the evolution of vorticity, the time consumed increases exponentially with every shedding of particles and hence it was concluded that it is not a viable option to choose to simulate the case of the full scale wind turbine using the direct method to solve the vorticity equation. However, if we choose to reduce the tip-speed ratio, it can be made certain that the wake will reach the required distance downstream to be able to capture the velocity profiles up to at least 5D. Scaling the radius alone allows us to simulate the wind turbine wake upto the required extent corresponding to the scaled radius and doing so reduces the tip-speed ratio as the RPM and wind speed are kept constant. As the focus, here in this thesis, is to demonstrate the capability of the solver to combine the actuator line model and the vortex particle scheme to study the wake, scaling the radius alone would prove conducive for this purpose. The new conditions as per this consideration are as in table 6.3

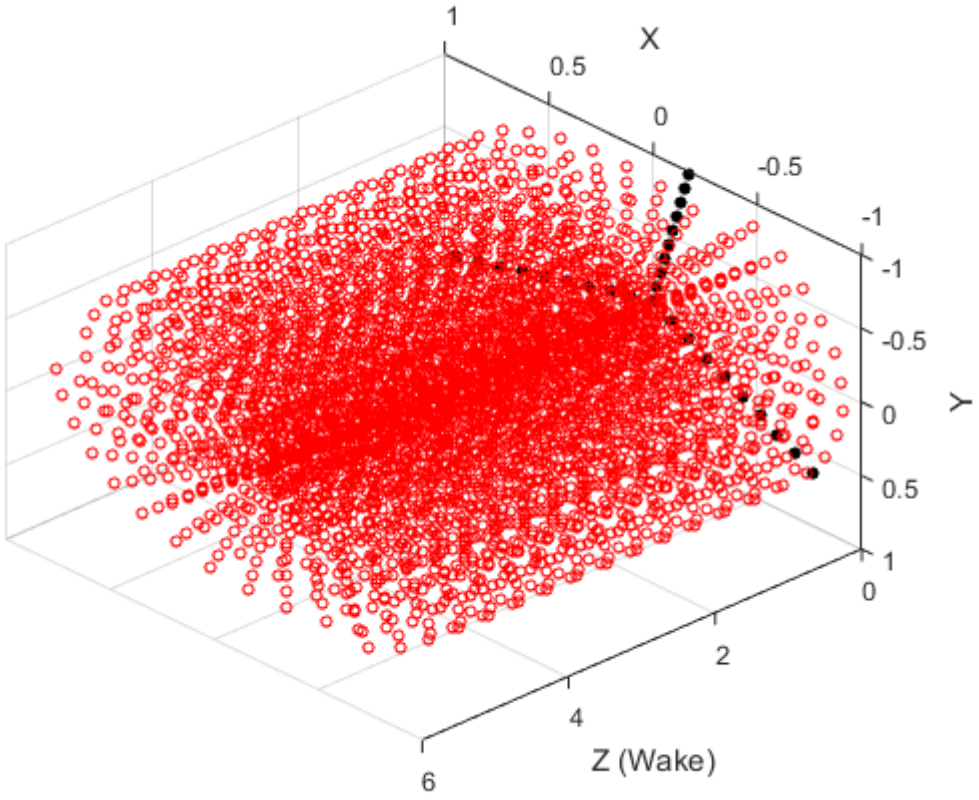


Figure 6.3: Wake generated with non-dimensional parameters

	Radius (m)	RPM	Wind speed (m/s)	Tip-speed ratio (derived)
WT VPM solver	1.0	70	7	1.04
Conditions from [13]	5.030	72	6.7	5.417

Table 6.3: New input conditions for the solver

6.1.5. Results pertaining to scaled radius alone

The blade was split into 10 sections and the circulation strength at these 10 sections were determined. This can be noted to be as shown in figure 6.5. The simple helical wake, for one blade, was generated without the convection and evolution of vorticities enabled. the corresponding images are as shown below in figure 6.6a.

It can be observed from 6.6a that the basic coding to generate a wake with the shed particles simply moving downstream without any induced velocities generates a perfectly helical wake as expected. The direction of vorticities is as shown in figure 6.6b and figure 6.6c. The vorticities of the particles are so oriented that they may simulate the direction of vorticity as in the trailing vortices at every section on the blade. These vortices are oriented in the 3-D space with their Z component oriented in the effective direction of incoming flow. The following figures 6.11 show the evolution of the wake with VPM enabled.

The cut-off radius was varied from 0.05 to 0.25 to observe its effects on the evolution of the wake. This is as depicted in the figures in figure 6.11b. The corresponding normalized velocity profile of the wake at a distance of 2D and 5D downstream are as depicted in the figures 6.10. The results are discussed for both cases.

6.1.6. Discussion of the results pertaining to scaled radius alone

It can be observed from 6.6a that the basic coding to generate a wake with the shed particles simply moving downstream without any induced velocities generates a perfectly helical wake as expected. The direction of

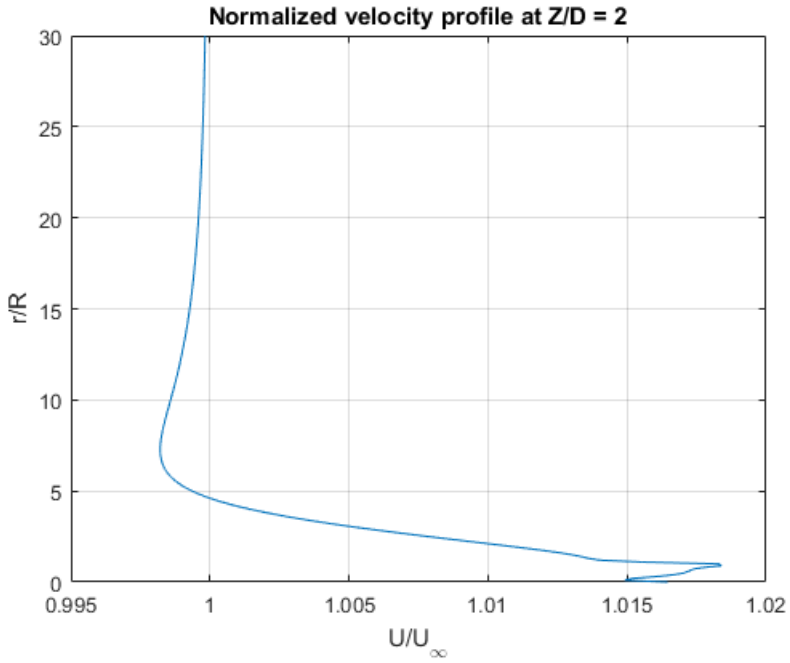


Figure 6.4: Erroneous wake velocity profile generated with non-dimensional parameters

vorticities is as shown in figure 6.6c.

Enabling the VPM segment in the code generates wake as seen in figure 6.11. In figure 6.11 it can be seen that the particles do not exhibit much disturbance in the flow that we would expect to see due to induction. It is quite possible that the magnitude of the vorticities assigned to the particles is insufficient. However, the expansion of the wake can be observed to have happened by noting the increasing radius of the wake in figure 6.10. The radial distance at which the wake velocity reaches the velocity of the ambient air can be observed to be increasing which effectively means that the radius of the wake itself is expanding.

To observe the capability of the solver we compare the velocity obtained with the normalized velocity profiles obtained for a wind turbine in [17]. It can be observed that the normalized wake velocity profile follows the same shape as that in the [17]. Quantitatively comparing them (although a quantitative comparison is not a wise option) we see that there is a discrepancy of approximately 40% with the observations made at $\frac{Z}{D} = 2$ for a cut-off radius of 0.15 (Error is calculated as in equation 6.7). This discrepancy can be seen to have increased to 98% at downstream location $\frac{Z}{D} = 5$. A cut-off radius of 0.15 has been observed to represent the NREL results closely.

$$\%error = \frac{(\frac{U}{U_\infty}) - ((\frac{U}{U_{ref}})_{NREL})}{((\frac{U}{U_{ref}})_{NREL})} \times 100 \tag{6.7}$$

This means that the model that has been implemented - Actuator line model for the blade combined with vortex particle scheme for the wake evolution - is capable of producing results that show the recovery of wake velocity at downstream locations. However, the results deviate by a very large magnitude with respect to the results from NREL simulations. These results have been produced after initializing the particle strength in the wake with a non-dimensional number that weighs the strength of the particles that have been shed. This also means that the strengths assigned to the shed particles are methodically correct.
To obtain improved results in the far wake (≥5D, wake shown in figure 6.8) the assignment of the strength of the particles is changed to taking into account the distance each particle would have been shed through at every time-step. For every shedding, or for every time-step, the particles travel by $0.14m$. For the wake to reach at least 5D, 100 sheds of particles were created. Referring to the figure 6.1, we see that at every time step the distance the tip vortex particle travels is much larger than that of the particle close to the root. For this

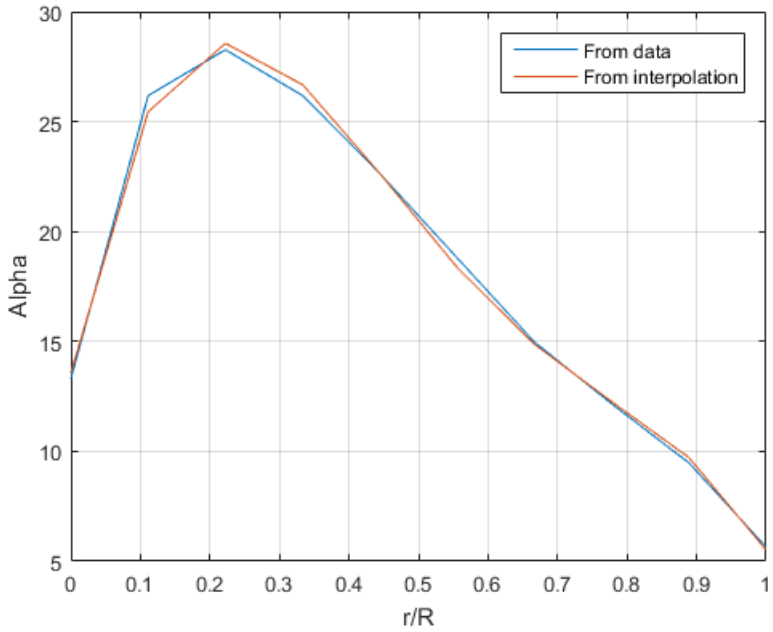


Figure 6.5: Circulation distribution from root to tip of blade

purpose, we take into account the length of the effective velocity vector (from figure 6.2) in equations 6.4-6.6. The new equations become as in equations 6.8-6.10.

$$\alpha_x = (dt\, U_{eff}/c)\,(\frac{d\Gamma}{dy}dy)\,\sin(\theta)\,\sin(\beta) \tag{6.8}$$

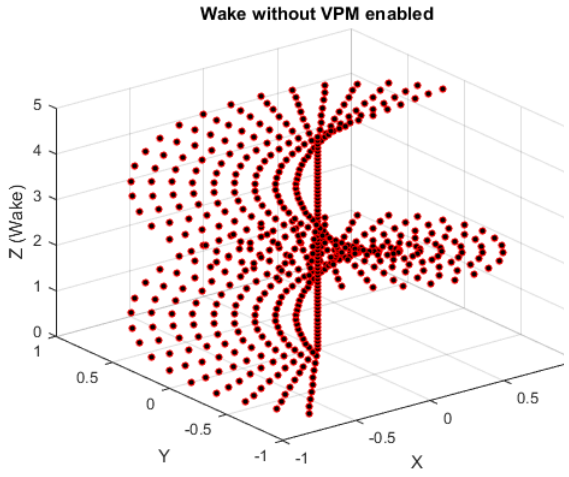
$$\alpha_y = -(dt\, U_{eff}/c)(\frac{d\Gamma}{dy}dy)\,\cos(\theta)\,\sin(\beta) \tag{6.9}$$

$$\alpha_z = -(dt\, U_{eff}/c)(\frac{d\Gamma}{dy}dy)\,\cos(\beta) \tag{6.10}$$

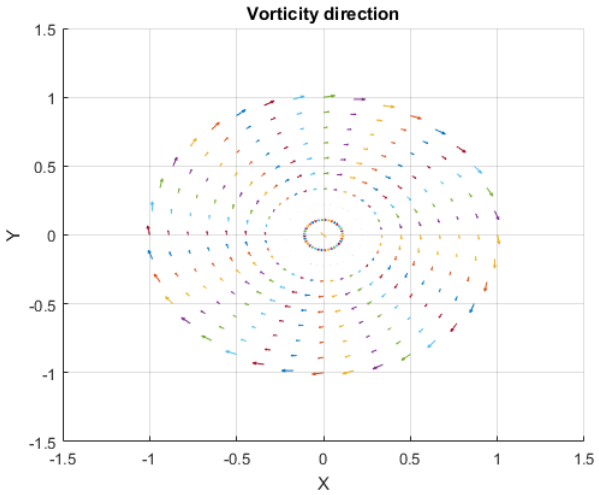
The results after implementing this equation are as shown in figure 6.9. Comparing figure 6.9a with the figure representing the profile at $\frac{z}{D} = 2$ in 6.10c and using equation 6.7, the %error for $\frac{z}{D} = 2$ has now reduced to 25% from 40%. However there is only a very small improvement in the result for the case of $\frac{z}{D} = 5$ which is around 2% lower from the earlier case.

6.1.7. Wake expansion

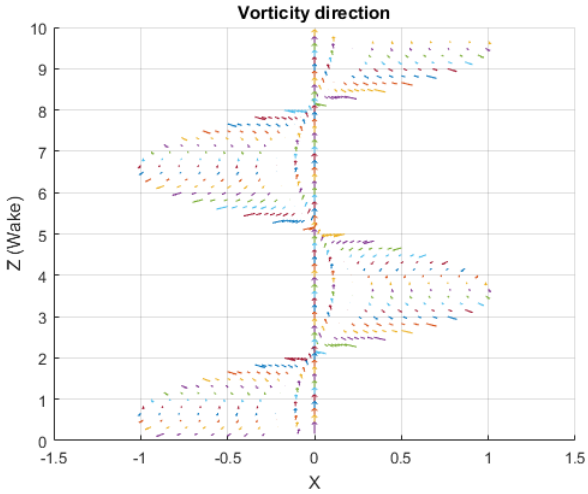
The expansion of the wake is not apparent in the plot that shows the evolved wake behind the rotor 6.11. However, the expansion of the wake can be studied by observing the distance from the root of the blade towards the tip where the normalized velocity reaches a magnitude of 1. By plotting the normalized velocity curve for varying distances downstream of the wind turbine we can observe the phenomenon of the wake expanding, as shown in figure 6.10. The plots are generated for the same positions downstream as observed in the NREL CFD experiment. The normalized velocity at $\frac{Z}{D} = 2$, in comparison to the CFD results, has deviated approximately by 40% as mentioned earlier. This is on account of the method used to assign the particle strengths at the first shedding of trailing vortices which then evolve downstream over time. The radial location where the normalized velocity reaches 1 can be observed to be approximately at a location greater than in the CFD results. The radius of the wake can be seen to be increasing from approximately $\frac{r}{R} = 2.5$ at $\frac{Z}{D} = 2$ to approximately $\frac{r}{R} = 5$ at $\frac{Z}{D} = 5$ while the same can be observed in the reference plots to be increasing from approxmately $\frac{r}{R} = 1$ at $\frac{Z}{D} = 2$ to approximately $\frac{r}{R} = 1.5$ at $\frac{Z}{D} = 5$. From these observations it is inferred that the VPM causes a very quick recovery of the wake over a span of 3D. It could be possible that the diffusion effects from the vorticity equation are quite large and that it is causing the rapid velocity recovery.



(a) A perfectly helical wake obtained with VPM disabled



(b) Vorticity direction of the particles in the wake



(c) Vorticity direction of the particles in the wake

Figure 6.6: Wake without VPM

6.1.8. Effect of varying the cut-off radius of the kernel function

From varying the cut-off radius to study its effects on the wake, from figure 6.11b, it can be observed that the velocity of the particles in the wake exhibit signs of decreasing levels of induction. At a location of $\frac{Z}{D} = 5$ downstream the velocity of the wake at the root of the blade recovers its velocity with increasing cut-off radius. At the same location the contraction of the radius of the wake is also evident with increasing cut-off radius. This is on account of the fact that with larger cut-off radius the influence on the particles in the near wake is reduced. The evolution of this low-influenced wake that recovers downstream has even lower normalized velocity. This is the effect that is seen at 5D.

6.1.9. Conclusion

A wind turbine wake solver has been developed using VPM and an attempt at validating the results has been made with improved results from an optimized formulation for assigning particle strengths. The blade was discretized using actuator line model with the circulation of particles on the actuator line obtained from the lift distribution of the blade using an immersed lifting line theory. At the discrete locations on the blade the circulations of particles was obtained from the Kutta-Jukowski theorem. These particles were allowed to be shed in the wake of the blade as the blade was rotated through angles of 10°. The initial simulation using non-dimensionalized paramerters proved to generate erroneous results while using the full scale blade geometry

as has been explained to be computationally expensive in MATLAB. It has been found that, for the case of scaled radius alone, the normalized velocity profile for the wake closely represents the result, for 2D, of a conventional wind turbine as presented in [17] for a cut-off radius of 0.15 with an error of 25%. However, it has been demonstrated that the results can be improved significantly by implementing the right formulation for the particle strengths in the wake. Therefore, it is concluded that it is quite possible to get accurate results, with respect to the results from NREL, when the right formulation for the particle strength is employed along with the full scale model of the blade. This formulation is not surely known at the time of this writing. At this point it stands that the developed solver cannot be validated quantitatively due to limitations in computational capacity to use the direct method. The developed solver, for the full scale turbine blade, is expected to produce accurate results when implemented in BBFMM3D but this is out of the scope of this thesis. If the results can be produced with significant accuracy then the combination of an actuator line model of the wind turbine blade with a vortex particle scheme would prove to be a valuable tool to visualize the flow in the wake.

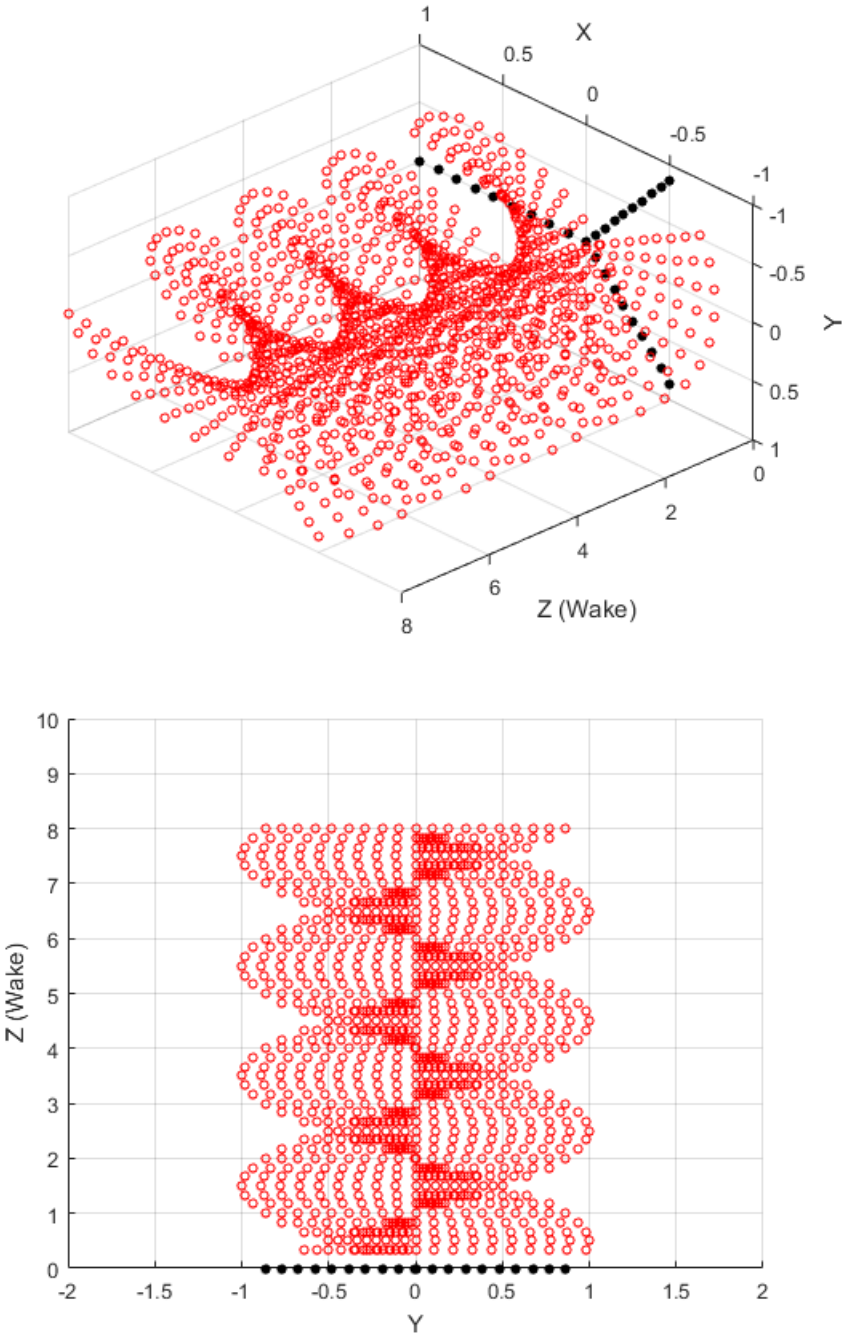


Figure 6.7: Wake with VPM

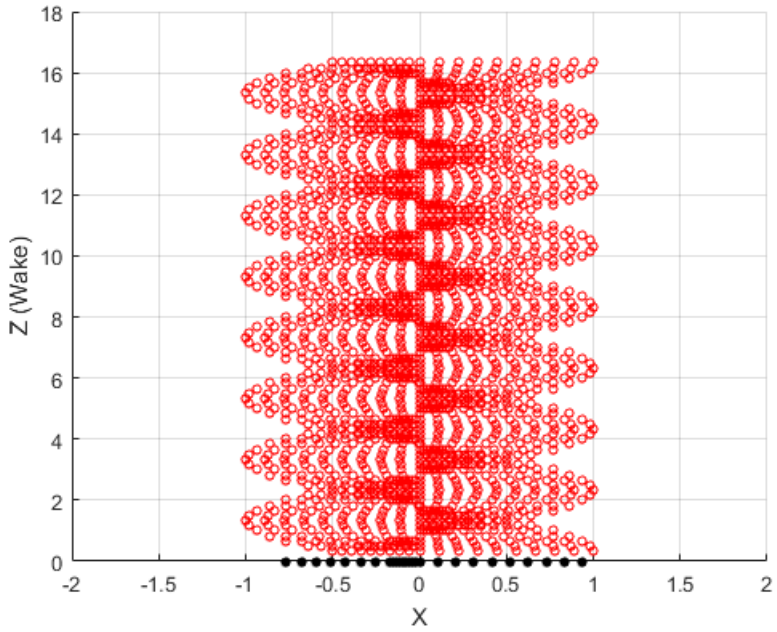
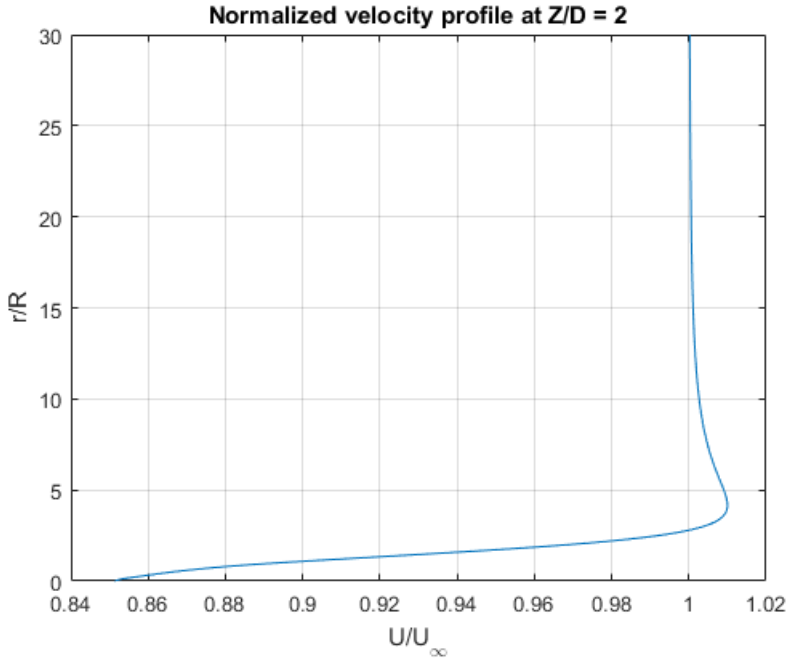
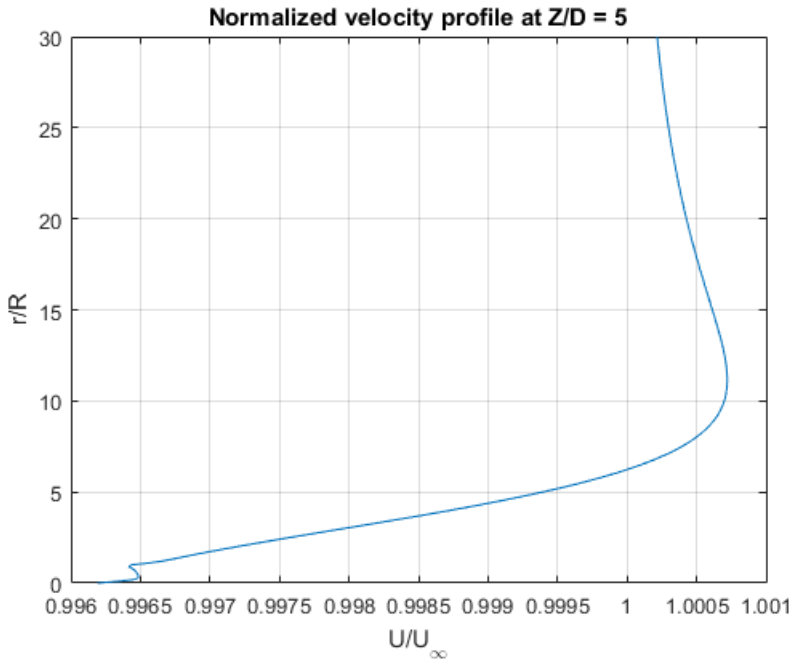


Figure 6.8: Wake that has reached at least 5D

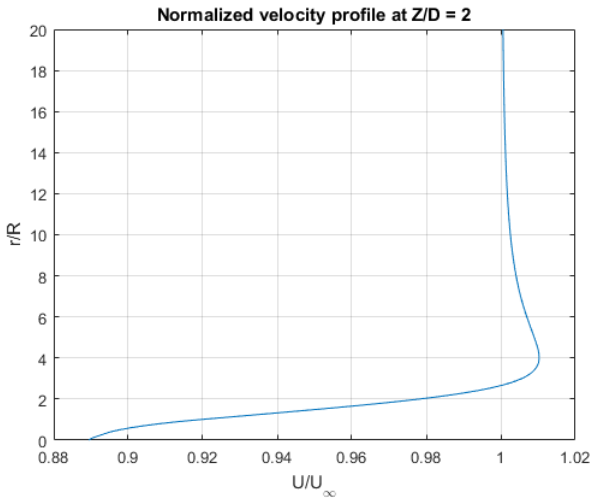


(a) Velocity profile at $\frac{z}{D} = 2$

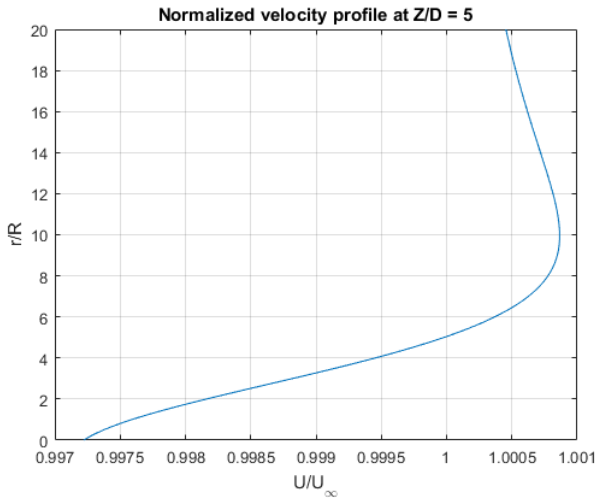


(b) Velocity profile at $\frac{z}{D} = 5$

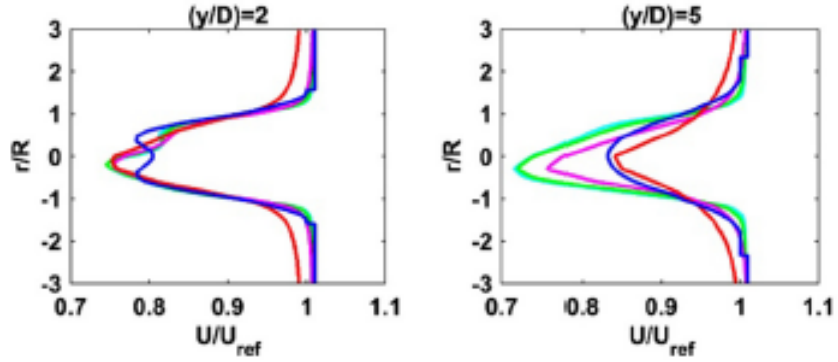
Figure 6.9: Optimized wake velocity profiles



(a) z/D = 2

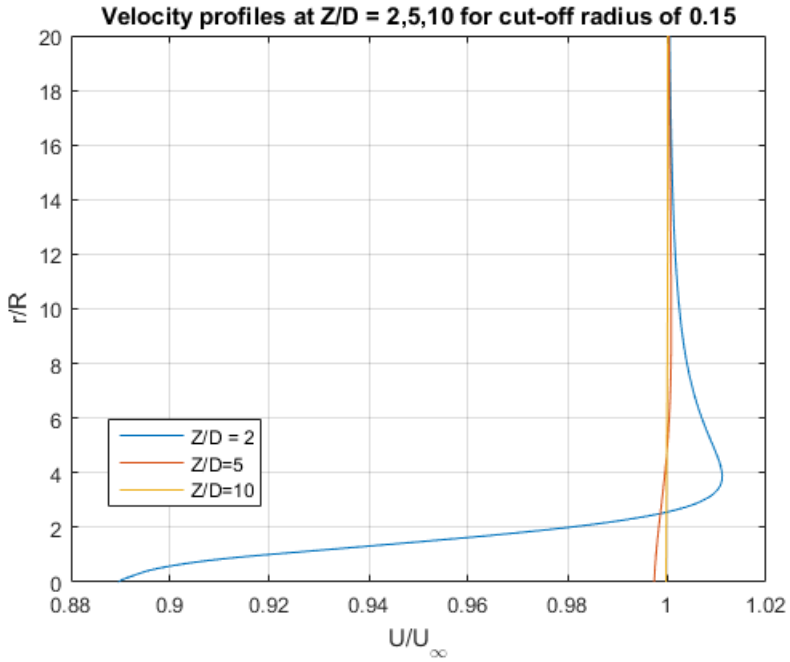


(b) z/D = 5

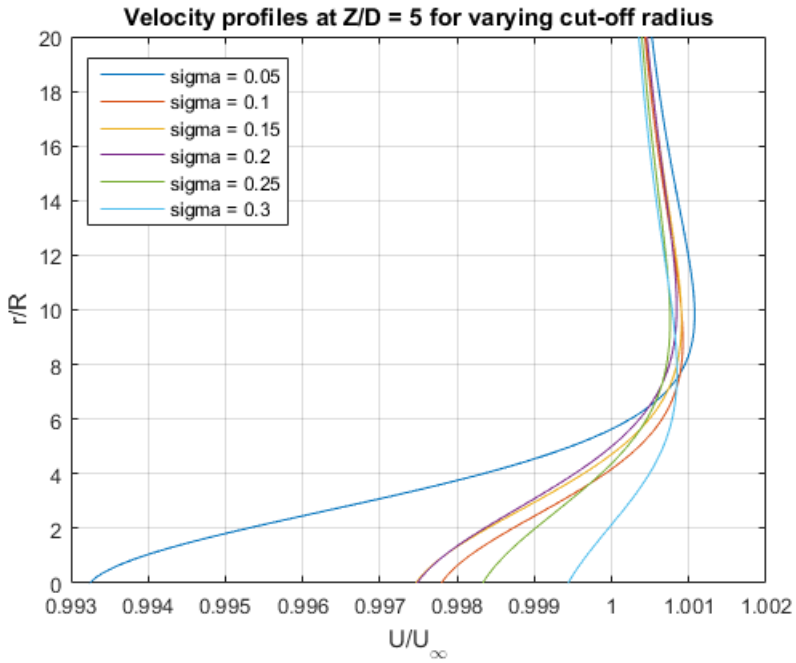


(c) Reference plot from NREL CFD experiments at [17],(Blue curve)

Figure 6.10: Wake normalized velocity profiles for increasing downstream distance



(a) Velocity profiles at increasing downstream locations for a cut off radius of 0.15



(b) Wake velocity profiles for increasing cut-off radius of the low-order algebraic kernel used

Figure 6.11: Wake velocity profiles for increasing downstream location and increasing cut-off radius

7

Conclusions and recommendations

The tools necessary to develop a full wind farm solver have been developed and demonstrated to be producing considerably agreeable results. Chapter 1, 2 and 3 dealt with an introduction to this research, literature survey performed to understand the methods of implementing vortex particle schemes for wake evolution and a theory of vortex particle method along with its algorithm was provided in chapter 3.

Chapter 4 provided a brief background to the Fast Multipole Method and demonstrated the validity of the FMM library, BBFMM3D. In this chapter the accuracy of the library was also observed to be of $O(10^{-4})$. Then, an explanation to what consequences in relative error occurred upon varying the length and number of particles was studied. A relation between the length of a cell and the level of the computational domain was also obtained in the process to understand the limit for the length that produced errors while FMM was in effect in the far field interactions. It was also clearly demonstrated that the time consumed for the FMM computation is significantly lower in comparison to that of the exact computation.

In chapter 5, a VPM scheme was validated for the case of vortex rings (single ring, Leap frogging of two rings, collision of two rings). This scheme is as developed by Winckelmans [35] using the particle strength exchange scheme for diffusion. The VPM scheme was successfully validated by presenting the decaying effect of the induced velocity term with increasing core radius with almost 100% accuracy with the results obtained in [35] by using a low-order algebraic kernel to compute the induced velocity. Using this valid code generated to simulate the single ring, cases of collision of two rings and the leapfrogging of two rings were also simulated to validate the diffusion terms and vortex stretching terms in the vorticity equation respectively. The case of single vortex ring was also simulated using BBFMM3D. It was observed the BBFMM3D produced agreeable results.

With a working VPM scheme developed in chapter 5, a wind turbine wake was simulated using this scheme in chapter 6. It was observed that the wake generated in this case qualitatively agrees with the physical model as it has been able to generate an expanding wake. The wake velocity profiles were generated at distance of approximately 2D, and 5D and it was found that the results were agreeable for a cut-off radius of 0.15 at a location 2D downstream from the blade. The accuracy of the velocity near the root of the blade was calculated to be approximately 85% with the optimized method to determine particle strengths. Although the expansion of the wake could not be observed in the simulated wake plot, it could be observed in the series of velocity profiles generated for the wake at increasing downstream positions by observing the radial distance from the root of the blade up to the point where normalized velocity reaches a value of 1. The cut-off radius that gives closely valid results, when compared with [17], has been determined to be 0.15.

This thesis has combined the actuator line model of a wind turbine with a vortex particle scheme to simulate the wake of a HAWT albeit without visible wake expansion. It was found that non-dimensionalizing the input parameters produces erroneous results which led to discarding the whole non-dimensionlizing procedure. It was then decided to use the parameter is their dimensional forms as available in [17]. The initial results obtained with a rudimentary weight for the particle strength in the wake proved to be quite erroneous. An optimized formulation that takes into account the length of the segment that a particle travels through with

every time-step has demonstrated that the results can indeed be improved significantly with the right formulation employed for the particle strength. However, at this point, the model used to simulate the wind turbine could not be validated with literature although the VPM scheme generated was perfectly validated against the results obtained in [35].

7.1. Recommendations

In order to validate the simulation with the results from literature, [17], it is recommended that the simulation is run using BBFMM3D as employing the FMM scheme in BBFMM3D would significantly accelerate the computation. This is to be accomplished using the full-scale model of the wind turbine. As BBFMM3D is meant to accelerate the n-body problem, with a valid code for the wind turbine, the simulation can be run for at least 15000 particles in the domain. This will also enable us to study the far-wake.

In this thesis, as has been shown, the low-order algebraic kernel has been used to compute the induced velocity due to vortex particles. This was chosen so as to validate the kernels and the algorithm being developed for the simulation that used an actuator line model. However, a new kernel has been shown to have been developed in the [35] which has the characteristics of a high-order algebraic kernel. The benefit of this kernel, apart from it being of high order, is the fact that it is strongly Gaussian-like. In order to study the working of the BBFMM3D wake solver, it is also recommended to implement this scheme by only shedding particles after, say, ten time steps.

It was also pointed out that the wake expansion was not visible in the wake plot. To be able to visualize the wake it is recommended that an appropriate formulation to determine the magnitude and orientation of the vorticity in the wake be developed. It could also possibly be fixed by increasing the number of particles to model the actuator line.

Bibliography

[1] Lorena A Barba. *Vortex Method for computing high-Reynolds number flows: Increased accuracy with a fully mesh-less formulation*. PhD thesis, California Institute of Technology, 2004.

[2] J Thomas Beale and Andrew Majda. Vortex methods. i. convergence in three dimensions. *Mathematics of Computation*, 39(159):1–27, 1982.

[3] E Branlard, P Mercier, Ewan Machefaux, Mac Gaunaa, and S Voutsinas. Impact of a wind turbine on turbulence: Un-freezing turbulence by means of a simple vortex particle approach. *Journal of Wind Engineering and Industrial Aerodynamics*, 151:37–47, 2016.

[4] Emmanuel Simon Pierre Branlard. *Analysis of wind turbine aerodynamics and aeroelasticity using vortex-based methods*. PhD thesis, DTU Wind Energy, 2015.

[5] Jacob S Calabretta. A three dimensional vortex particle-panel code for modeling propeller-airframe interaction. 2010.

[6] Philippe Chatelain, Stéphane Backaert, Grégoire Winckelmans, and Stefan Kern. Large eddy simulation of wind turbine wakes. *Flow, turbulence and combustion*, 91(3):587–605, 2013.

[7] Alexandre Joel Chorin. Numerical study of slightly viscous flow. *Journal of Fluid Mechanics*, 57(4):785–796, 1973.

[8] Georges-Henri Cottet and Petros D Koumoutsakos. *Vortex methods: theory and practice*. Cambridge university press, 2000.

[9] JB de Vaal, Martin Otto Laver Hansen, and T Moan. Validation of a vortex ring wake model suited for aeroelastic simulations of floating wind turbines. In *Journal of Physics: Conference Series*, volume 555, page 012025. IOP Publishing, 2014.

[10] Pierre Degond and S Mas-Gallic. The weighted particle method for convection-diffusion equations. i. the case of an isotropic viscosity. *Mathematics of computation*, 53(188):485–507, 1989.

[11] Victor Montague Falkner. The calculation of aerodynamic loading on surfaces of any shape. Technical report, AERONAUTICAL RESEARCH COUNCIL LONDON (UNITED KINGDOM), 1943.

[12] William Fong and Eric Darve. The black-box fast multipole method. *Journal of Computational Physics*, 228(23):8712–8725, 2009.

[13] P Giguere and MS Selig. Design of a tapered and twisted blade for the nrel combined experiment rotor. Technical report, National Renewable Energy Lab., Golden, CO (US), 1999.

[14] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.

[15] Chengjian He and Jinggen Zhao. Modeling rotor wake dynamics with viscous vortex particle method. *AIAA journal*, 47(4):902, 2009.

[16] Doug Hunsaker and Warren F Phillips. A numerical lifting-line method using horseshoe vortex sheets. 2011.

[17] Ijaz Fazil Syed Ahmed Kabir and EYK Ng. Effect of different atmospheric boundary layers on the wake characteristics of nrel phase vi wind turbine. *Renewable energy*, 130:1185–1197, 2019.

[18] I Katic, J Højstrup, and Niels Otto Jensen. A simple model for cluster efficiency. In *European wind energy association conference and exhibition*, pages 407–410, 1986.

[19] Kunio Kuwahara and Hideo Takami. Numerical studies of two-dimensional vortex motion by a system of point vortices. *Journal of the Physical Society of Japan*, 34(1):247–253, 1973.

[20] Anthony Leonard. Vortex methods for flow simulation. *Journal of Computational Physics*, 37(3):289–335, 1980.

[21] PBS Lissaman. Energy effectiveness of arbitrary arrays of wind turbines. *J. Energy*, 3(6):323–328, 1979.

[22] Evan H Martin. *Assessment of panel and vortex particle methods for the modelling of stationary propeller wake wash*. PhD thesis, Memorial University of Newfoundland, 2015.

[23] Brian Maskew. Program vsaero theory document: a computer program for calculating nonlinear aerodynamic characteristics of arbitrary configurations. 1987.

[24] S McTavish, S Rodrigue, D Feszty, and F Nitzsche. An investigation of in-field blockage effects in closely spaced lateral wind farm configurations. *Wind Energy*, 18(11):1989–2011, 2015.

[25] Daniel G Opoku, Dimitris G Triantos, Fred Nitzsche, and Spyros G Voutsinas. Rotorcraft aerodynamic and aeroacoustic modelling using vortex particle methods. In *23rd International Congress of Aeronautical Sciences (ICAS 2002), Toronto, ON, Canada, Sept*, pages 8–13, 2002.

[26] Søren Ott, Jacob Berg, and Morten Nielsen. Linearised cfd models for wakes. 2011.

[27] L Rosenhead. The formation of vortices from a surface of discontinuity. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 134(823):170–192, 1931.

[28] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 910–914. ACM, 2005.

[29] Jens Norkær Sorensen and Wen Zhong Shen. Numerical modeling of wind turbine wakes. *Journal of fluids engineering*, 124(2):393–399, 2002.

[30] Philippe R Spalart. Numerical simulation of separated flows. 1983.

[31] SG Voutsinas, KG Rados, and A Zervos. The effect of the non-uniformity of the wind velocity field in the optimal design of wind parks. In *Proceedings of the 1990 European Community Wind Energy Conference, Madrid, Spain*, pages 181–5, 1990.

[32] SG Voutsinas, KG Rados, and A Zervos. Wake effects in wind parks. a new modelling approach. In *Proc. European Community Wind Energy Conf, Lübeck-Travemünde, Germany*, pages 444–447, 1993.

[33] Spyros G Voutsinas. Vortex methods in aeronautics: how to make things work. *International Journal of Computational Fluid Dynamics*, 20(1):3–18, 2006.

[34] David J Willis, Jaime Peraire, and Jacob K White. A combined pfft-multipole tree code, unsteady panel method with vortex particle wakes. *International Journal for numerical methods in fluids*, 53(8):1399–1422, 2007.

[35] Gregoire Stephane Winckelmans. *Topics in vortex methods for the computation of three-and two-dimensional incompressible unsteady flows*. PhD thesis, California Institute of Technology, 1989.

[36] GS Winckelmans and Anthony Leonard. Contributions to vortex particle methods for the computation of three-dimensional incompressible unsteady flows. *Journal of Computational Physics*, 109(2):247–273, 1993.

[37] Ziying Yu, Xing Zheng, and Qingwei Ma. Study on actuator line modeling of two nrel 5-mw wind turbine wakes. *Applied Sciences*, 8(3):434, 2018.

[38] A Zervos, S Huberson, and A Hemon. Three-dimensional free wake calculation of wind turbine wakes. *Journal of Wind Engineering and Industrial Aerodynamics*, 27(1-3):65–76, 1988.