Resolution and Boundary Placement Effects on Discontinuous Galerkin Simulations of Counter-rotating Vortex Interaction in a Confined Domain

by

Rachel J. Reitz

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 14, 2013

Keywords: discontinuous Galerkin method, computational fluid dynamics, two-dimensional vortex interaction

Copyright 2013 by Rachel J. Reitz

Approved by

Andrew Shelton, Chair, Assistant Professor of Aerospace Engineering
Roy Hartfield, Woltosz Professor of Aerospace Engineering
Brian Thurow, Reed Associate Professor of Aerospace Engineering
George Flowers, Professor of Mechanical Engineering, Dean of Graduate School

Abstract

The wakes of fixed-wing and rotary-wing aircraft present some of the most complex and challenging problems for numerical simulation, as fully capturing the interaction of the various vortices and coherent structures requires immense computational power and time. With current methods, the cost of performing such simulations is prohibitive or even impossible, so it is thought that a more advanced and efficient algorithm is needed to make such investigations feasible *now*, rather than waiting for computational power to increase over time. One promising option that has the potential to provide such efficiency is the discontinuous Galerkin method.

The discontinuous Galerkin method is a locally conservative, robust, flexible, and high-order accurate numerical scheme. It is relatively new to applications in the field of computational fluid dynamics, so its sensitivity to certain parameter changes is not thoroughly documented. This work uses a test case of two-dimensional vortex interaction to catalogue the effects of varying resolution through the use of polynomial degree and cell density. The vortices are confined to a box with reflective boundaries, so the effect of moving the boundaries closer or further is also investigated while applying a new initialization to ensure an initial condition that is consistent with the boundary conditions. Changes are noted by considering comparisons of vortex trajectory and kinetic energy error for each case. The results present data for polynomials as high as 14^{th} order and confirm that the method not only works with large cell sizes, but actually excels.

Acknowledgments

First, I would like to thank Auburn University for providing such a wonderful environment in which to complete my graduate work. I would also like to thank the Aerospace department and my committee for their continued patience, guidance, and support during the last three years. I owe a special thanks to Dr. Shelton, whose incredible depth of knowledge is inspiring, for being both kind and patient enough to pass on some small part of that knowledge to me. I hope I use it well and make you proud.

I also could not have successfully made the transition from physics to aerospace without my fellow graduate students, who were always so willing to help when I was struggling and provide conversation when I needed a break. You are the heart and soul of our department and it's been great getting to know all of you.

Finally, to my family, thank you for always being so proud and supportive of whatever I set my sights on. These have been the most difficult but rewarding years so far, and I am grateful for the encouragement and understanding from everyone. To Brian, thank you for making my time here (and wherever else we may end up) so enjoyable. I truly couldn't have done it without you.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As more and more complex problems are requiring simulation, more efficient ways to do so are being sought to bridge the gap between the computational power required and the power that is actually available. While it is true that technology and computing capability doubles approximately every two years, the size of problems and the capability they require vastly exceeds that rate and the gap is only widening. Take, for instance, the flow field behind a heavy transport aircraft in a landing configuration, or the self-interacting wake of a rotorcraft. These are highly complex flow fields with many vortices constantly interacting, growing, and ultimately deteriorating at different rates. However, fully simulating a problem that large and complicated requires immense time and other resources that render it impractical to model in its entirety using currently available numerical methods.

One solution to this problem is to develop a more advanced algorithm that incorporates the best aspects of some of the traditional methods, such as geometric flexibility, local approximation, high order, and of course, speed. Then, this method can be used to more efficiently simulate and accurately capture some of the dynamics in three-dimensional flow that until now have been impossible or impractical to capture numerically. One relatively new method that seems to fulfill all of these needs is the discontinuous Galerkin method. However, with a newer method, there are invariably questions that arise about accuracy and discoveries to be made about how it performs under certain circumstances, since it lacks the documentation of other more well-established methods.

Therefore, before being able to jump into the ultra large-scale problems suited for a more efficient method, these issues must be addressed. With that in mind, this work seeks

to apply the discontinuous Galerkin method to the older and more fundamental problem of two-dimensional vortex interaction with the goal of building towards modeling complex aircraft wakes. Standard practice for modeling confined vortices disregards the effects of reflective boundaries and presents initial conditions that do not match the physics implied by the boundary conditions. Therefore, this work simultaneously aims to rectify the disagreement by proposing an alternate initialization procedure. By studying the core parameters of the discontinuous Galerkin method in a more controlled and computationally cheaper environment, it is hoped that their effect on the method can be better understood. With the foundation laid, that knowledge can then be put to use and the most efficient combination of parameters can be confidently implemented in the more complex cases.

1.2 Background

Since this work is rooted in aircraft wakes, vortex interaction, and the discontinuous Galerkin method, a brief overview of each follows in order to provide context. Vortices and vortex interaction have been topics of study for decades, due to their appearance in a wide variety of naturally-occurring phenomena. The study of coherent structures in turbulent flows has led to a hope that a better understanding of vortices will give meaning and pattern to the seemingly random motion of turbulence [1]. Single vortices of the isentropic variety also serve as an excellent test case for numerical studies, as they should simply convect with the free-stream velocity, and any decay or deviation from this can be attributed to the numerical scheme.

When it comes to modeling pairs of vortices, the work is usually framed in the context of aircraft wake, and especially so when the pair are counter-rotating. Aircraft separation at airports across the globe is one of the most practical problems that has yet to be addressed very effectively, so any better understanding of the physics occurring in aircraft wake that can lead to a safe reduction in take-off and landing separation between aircraft is valued. In a 1998 paper, Philippe Spalart outlined some of both the analytical and experimental

work that had been done at that point to contribute to aircraft wake knowledge [2]. He pointed out some of the limitations of each, admitting that numerical simulations made too many assumptions to really be useful and that wind tunnel tests provided data of a smaller scope than is ideal. On the other hand, full-scale flight testing was cost-prohibitive and could be limited as well if only commercial flights were considered in order to save on cost. With these factors considered, he conceded that more work was needed, but also admitted that a truly useful wake model would have to account for various atmospheric conditions in addition to the vortex dynamics in order to really improve upon the current system. For these reasons, the same general spacing rules based on weight class have remained in use since the publication of his paper.

However, more concerted efforts have since been made in the United States and abroad with programs specifically dedicated to understanding aircraft wake and ultimately reducing aircraft spacing. European research programs include Fundamental Research on Aircraft Wake Phenomena (FAR-Wake) and Wake Vortex Characterization and Control (C-WAKE). FAR-Wake was a three-year program from 2005-2008 that involved collaboration between 17 organizations in 8 countries, and investigated vortex instabilities and decay, along with vortex interactions with jets and wakes [3]. They employed a multi-faceted approach that combined experimental, theoretical, and numerical methods to aid in wake characterization and prediction, with the long term goal of alleviating aircraft wakes once their behavior was better understood. Although most of their work centered on more simplified test cases, it produced new results concerned with the fundamental behavior of vortices.

In the United States, the primary wake research program known as Aircraft Vortex Spacing System (AVOSS) began in 1994 and involved six years of research by NASA that culminated in a demonstration at Dallas Fort Worth (DFW) airport in July of 2000 [4]. The idea behind AVOSS was that under Visual Flight Rules (VFR), pilots were given the freedom to control their own spacing, and oftentimes it was less than the prescribed spacing mandated under Instrument Flight Rules (IFR) conditions. This has lead to the general

belief that the FAA spacing requirements that rely on worst-case weight differences and long wake lifespans are overly conservative and commercial airports could benefit from a system such as AVOSS that recommends shorter (but still safe) separation distances. Therefore, the goal of AVOSS was to provide a real-time system that receives current meteorological conditions and exact aircraft type (not a broad weight class) and unifies it with known understanding of vortex convection and decay [5, 6]. It should be noted that the goal was *not* to model the actual wake, but to provide an estimated range of the wake so that it could be avoided. AVOSS proved at least moderately successful, as during its implementation at DFW in 2000, it resulted in a theoretical 6% average increase (up to a maximum of 16%) in throughput under IFR conditions. The success of AVOSS lead to further efforts, like a joint FAA/NASA program and a project at NASA Ames called Wake Vortex Avoidance System (WakeVAS).

Other more individual efforts toward vortex understanding include those by Winckelmans [7] and Chatelain [8], who both employed vortex methods to study counter rotating vortices in the context of aircraft wakes. Specifically, Winckelmans is a proponent of the Lagrangian methods (mainly particle methods) for the negligible dispersion error and conservation of energy. He looked at both 2-D and 3-D interaction, utilizing particle redistribution to maintain accuracy. Chatelain also employed vortex particle methods, but focused on the implementation with parallel computers, which enabled the usage of well over a billion particles. Both researchers' main focus was accurately capturing the small-scale instabilities that arise in vortex interaction.

1.2.1 Towards a New Method

The work described above, whether experimental or numerical, involved immense cost and would certainly benefit from further development of CFD methods, but where does the most room for improvement exist? When considering schemes especially well-suited to CFD, some of the most-sought-after qualities relate to the "mesh," or discretization of the domain,

as oftentimes the solution is only as good as the mesh. For instance, a scheme that limits the cells to a certain shape or distribution would not be as helpful as one that can accommodate flexible or irregular cells. Another important quality is that the method be as accurate as possible without paying an enormous price with speed and efficiency. Lastly, the method should not be so computationally expensive that simulations are overly limited in size and scope or require gross estimations or assumptions.

It has already been suggested that the discontinuous Galerkin method delivers on all of the above qualities, but in the world of numerical simulation, there are already three very widely used approaches: the finite difference, finite element, and finite volume methods. Given their well-documented performance, why the need for a fourth method at all? Each method's key properties are outlined below and assessed according to their ability to meet all of the criteria for an ideal CFD scheme.

Finite Difference Method

The finite difference (FD) method is centered on replacing a partial derivative with an expression based on Taylor series expansions. The first step toward that expression is to define a relationship between the velocities at two points in the grid like in Equation 1.1, where $u_{i+1,j}$ is the x velocity at point $(i+1, j)$ and $u_{i,j}$ is defined likewise. The grid is assumed to be uniform and cartesian in this case, where Δx is the cell size.

$$u_{i+1,j} = u_{i,j} + \left(\frac{\partial u}{\partial x}\right)_{i,j} \Delta x + \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} \frac{(\Delta x)^2}{2} + \left(\frac{\partial^3 u}{\partial x^3}\right)_{i,j} \frac{(\Delta x)^3}{6} + ... \tag{1.1}$$

It should be noted that $u_{i+1,j}$ and $u_{i,j}$ are somewhat arbitrary selections, as they just must be two different points with Equation 1.1 adjusted accordingly. There are some common choices, and in some cases, the two points are neighbors, as in the choice of $u_{i-1,j}$ and $u_{i,j}$, but sometimes not, as in the choice of $u_{i-1,j}$ and $u_{i+1,j}$. The specific choice is what determines whether the method is a forward, backward, or central difference (Equation 1.1

demonstrates a forward difference, since only information to the right of gridpoint (i, j) is used). The distribution of points used in the scheme is called the 'stencil' and having many points in the stencil or a very wide stencil is typically more complicated (and expensive).

Since the real goal is to have an approximation for a partial derivative, Equation 1.1 is rearranged and solved for $\frac{\partial u}{\partial x}$ in order to get the equation needed:

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} - \left(\frac{\partial^2 u}{\partial x^2}\right)\frac{\Delta x}{2} - \left(\frac{\partial^3 u}{\partial x^3}\right)_{i,j}\frac{(\Delta x)^2}{6} - ... \tag{1.2}$$

Theoretically, Equation 1.2 would be exact if it had infinite terms. However, since infinite terms are impossible, unnecessary, and inefficient, Equation 1.2 must be truncated at some point. Any terms that still have partial derivatives in them are dropped, leaving just the "finite difference" portion, and the terms dropped constitute the truncation error associated with the method. Depending on the formulation, and whether the method is forward, backward, or central, the error is generally first or second order. For example, Equation 1.2 is an instance of first order error, since the first dropped term is on the order of Δx. While there is the potential for accuracy beyond second order with FD methods, it becomes increasingly more difficult due to the stencil breadth required and the increase in computation time to implement it. A broad stencil near the boundary is also an issue, as it requires data at points that don't exist.

In conclusion, the finite difference method is simple and fast, since the solution is built one point at a time. It is also beneficial that higher order accuracy (above second order) is achievable, but the cost is prohibitive. Since the error is a function of Δx, reducing the spacing between points is an easy and straightforward way to reduce error, but this also quickly increases computational cost. Also, the geometric flexibility is complicated by the local smoothness requirement and discontinuities are not allowed. All in all, finite difference is a good scheme, but is not ideal since it limits mesh properties.

Finite Element Method

In contrast to the point-defined FD method, the finite element method (FEM) is based on a domain that is represented by non-overlapping elements or cells. These elements are highly flexible and can easily adapt to complex geometries. However, the solution is not expressed locally, and is rather defined *globally* with piecewise polynomials on the cells. The use of polynomials allows for high order accuracy, but the implicit (global) formulation is expensive. Also, since the entire domain is solved at once (and not one node at a time like with finite difference), no inherent direction can be built in to simulate the physics of the flow. Incorporating information about the flow field is an important feature for fluid dynamics, so this especially makes it less desirable for CFD applications. Lastly, the global solution means that using more cells quickly increases the size of the matrices that have to be inverted during iterations to create the solution, so if a lot of cells are needed, it gets even more expensive. Thus, FEM would be much more useful for CFD if it were not globally defined.

Finite Volume Method

Perhaps the most popular of the three methods for applications in CFD is the finite volume (FV) method, as some version of it is at the foundation of many commercial codes. Instead of discretizing the partial differential form of the governing equations, the approach is to discretize the integral form of the equations. Much like with FEM, the domain is divided into cells, and these are also quite flexible and can handle complex geometries and unstructured meshes. The cells are centered on the grid points, so that the boundaries between cells lie half way between neighboring grid points. The solution is composed of cell averages (in other words, piecewise constant approximations), so discontinuities between cells with different values are addressed with numerical flux functions. The method is locally conservative and explicit in time (meaning that the solution is built cell by cell and is much cheaper than an implicit scheme). Due to the local nature, upwinding can be used to more

accurately simulate the physics of the flow if there is a certain direction to the problem (one of the things lacking with FEM). The FV method is quite a robust option and all of the benefits explain why it is so widely used in fluid dynamics. However, the piecewise constant representation of the solution is very low order and can make it hard to accurately capture certain flow features. The solution is to either decrease the cell size or find a way to increase the order of the method. However, attempts at higher order are complicated by large stencils and difficult boundary treatment and decreasing cell size small enough to capture small scales is very expensive.

Argument for a Fourth Method

When considering the desire for a method that is fast, flexible, and high-order accurate, each of the above methods fails in some regard. The elements of FD are not very flexible, FEM is implicit in time, and FV lacks high order capabilities. Thus, the ideal method would combine the flexible high-order elements of FEM with the local properties of FV, and this idea of merging the best qualities of the two into one method is how the discontinuous Galerkin finite element method (DG-FEM, or just DG for short) came about. By utilizing cells of high degree, instead of constant value, the cell count can be decreased (which lowers cost), and the explicit formulation makes it computationally fast and cheap. The discontinuity between cells persists, but the extensive history of numerical fluxes can be applied in a similar way as in FV.

The DG method was first proposed in a 1973 paper by Reed and Hill to solve the neutron transport equation on triangular elements [9]. Analyses were carried out by Lesaint and Raviart [10] in 1974 and again in 1986 by Johnson [11], but beyond that, the method was not used much at first. However, during the 1990's, gradual advancements were made in making the method more flexible and capable. The first attempts to extend the original DG method beyond linear problems and apply it to nonlinear problems initially led to implicit schemes, which are computationally expensive. In an effort to keep the method explicit

(cheap), the Runge-Kutta DG methods were developed. Next, the methods were extended to convection-diffusion systems, which spawned another type of DG method known as the local DG (LDG) method. Since then, there has been a surge in the use of DG methods for a wide variety of applications [12]. Yet, it is only relatively recently that the method has been considered for fluid dynamics use [13] [14].

As one of the more prolific researchers of the DG method, Cockburn wrote a paper outlining the method and its main properties [15]. In it, he discusses the implications of the method's discontinuous nature, which leads to block-diagonal matrices that enable it to be parallelizable. Not only does he show it to be parallelizable, but also to be extremely efficient in parallel, as even with 256 processors, total parallel efficiency remains above 97%. He also mentions that DG methods are especially well-suited for using with adaptive algorithms and are locally conservative, which is very important in fluid dynamics applications.

In another publication by Cockburn and Shu [16], they review the Runge-Kutta discontinuous Galerkin (RKDG) methods. They are used for non-linear convection-dominated problems and are popular in the fluid dynamics community. They uniquely merge a special class of Runge-Kutta time discretization with a finite element space discretization that incorporates numerical fluxes and slope limiters, which yields a scheme that is non-linearly stable regardless of accuracy. They show that high degree polynomials do not degrade the resolution of strong shocks and is more efficient on smooth regions.

Other work involving the DG methods includes that by Bassi and Rebay, who applied the method to numerically solve the compressible Navier-Stokes equations [17]. They began by extending a method initially intended for the Euler equations and treated the viscous terms with a mixed formulation. Performance was analyzed by considering laminar compressible flow on a flat plate as well as three conditions of flow around a NACA0012 airfoil. It was found that accurate solutions could be computed even on coarse meshes as long as higher degree elements were used.

1.3 Objective

Since much of the work in the literature addresses the small scale behavior of vortex interaction, including instabilities and turbulence modeling, this research does not seek to duplicate those results or expound on them. The focus will be less on the goal of exactly modeling a particular phenomenon and more on the observable differences that arise due to different settings. Thus, whereas most vortex research encompasses flows with a variety of Reynolds numbers, viscosities, and vortex strengths, this work will center on a single flow regime simulated using different combinations of case-defining parameters.

The discontinuous Galerkin method requires at least three fundamental parameters to be chosen before the simulation can even begin: polynomial degree, domain size, and cell density (essentially cell size). Considering that these three values can have a huge impact on both the accuracy and computational time of the method, their selection is not trivial. Thus, these three parameters will be systematically varied to discover how they impact the solution through analysis of error, kinetic energy decay, and vortex trajectory. The result should be a more thorough understanding of appropriate choices for a discontinuous Galerkin simulation, and how one can expect those choices to affect the error and computational time compared to other parameter options. Along the way, a different initialization process will also be developed to remedy the problem of inconsistent initial and boundary conditions when reflective boundaries are used.

Chapter 2

Formulation of the Method

As with many numerical methods, the first step in outlining the Discontinuous Galerkin (DG) method is to decide how to discretize the domain Ω into elements. The domain can be divided into K distinct elements D^k that do not overlap, but just touch and conform to the boundary, such that Ω is well-approximated by the elements:

$$\Omega \simeq \bigcup_{k=1}^{K} D^k \tag{2.1}$$

Then we say that the state $q(x,t)$ can be approximated by

$$q(x,t) \backsimeq q_h(x,t) = \bigoplus_{k=1}^{K} q_h^k(x,t) \tag{2.2}$$

where q_h denotes an approximate state that is made up of representations on the same K piecewise continuous elements (cells) referred to in Equation 2.1. Each $q_h^k(x,t)$ can be expressed with a basis

$$q_h^k(x,t) = \sum_{j=0}^{N} \tilde{q}_j^k(t) b_j^k(x) \tag{2.3}$$

where each $\tilde{q}_j^k$ is a coefficient, and each b_j^k is a basis function of some degree N to be determined later, but can be any of various polynomials or Fourier series. Essentially, the data is broken down by cells and represented on those cells by weighted functions, where the weights are determined by the value of the function at that point. This idea of expressing functions with bases is a central idea in the DG method and will come into play often.

In order to further explore the DG method, we can consider a generic conservation law for the state q (where $q = [\rho, \rho u, \rho e_t]$ in one dimension). Since no numerical method is exact,

we say that there must be a residual:

$$R_h^k(x,t) = \frac{\partial q_h^k}{\partial t} + \frac{\partial f(q)_h^k}{\partial x} \tag{2.4}$$

However, we can multiply each side by a (still unspecified) basis function and require the residual to vanish:

$$\int_{D^k} b_i^k \, R_h^k \, \mathrm{d}x = \int_{D^k} b_i^k \left(\frac{\partial q_h^k}{\partial t} + \frac{\partial f_h^k}{\partial x} \right) \mathrm{d}x = 0 \tag{2.5}$$

Taking only the right portion and expanding the expression, the vanishing residual requires

$$\int_{D^k} b_i^k \, \frac{\partial q_h^k}{\partial t} \, \mathrm{d}x \; + \int_{D^k} b_i^k \, \frac{\partial f_h^k}{\partial x} \, \mathrm{d}x = 0 \tag{2.6}$$

The second term in Equation 2.6 can be integrated by parts by realizing that $\dfrac{\partial}{\partial x}(bf) = \dfrac{\partial b}{\partial x} f + b \, \dfrac{\partial f}{\partial x}$ can be rearranged to give $b \, \dfrac{\partial f}{\partial x} = \dfrac{\partial}{\partial x}(bf) - \dfrac{\partial b}{\partial x} f$, which is the same form as the second term in Equation 2.6. Once applied, the result is

$$\int_{D^k} b_i^k \, \frac{\partial q_h^k}{\partial t} \, \mathrm{d}x + \int_{D^k} \frac{\partial}{\partial x}(b_i^k \, f_h^k) \mathrm{d}x - \int_{D^k} \frac{\partial b_i^k}{\partial x} f_h^k \, \mathrm{d}x = 0 \tag{2.7}$$

With the middle term simplified, it becomes

$$\int_{D^k} b_i^k \, \frac{\partial q_h^k}{\partial t} \, \mathrm{d}x + \int_{\partial D^k} b_i^k \, f_h^k \, n_x^k \, \mathrm{d}s \; - \int_{D^k} \frac{\partial b_i^k}{\partial x} f_h^k \, \mathrm{d}x = 0 \tag{2.8}$$

At this point, one might notice that the middle term of Equation 2.8 involves integrating around the edges ∂D^k of the element, which means using values at the interface between two neighboring elements and an outward facing normal n_x^k. This presents a problem because the elements are piecewise continuous, but not continuous across cell boundaries. Therefore, if a value is needed at the interface, which of the two prevails - the left or the right?

Figures 2.1 and 2.2 illustrate the predicament, where there are two distinctly different values at each location on the interface between the two cells. In the figures, the grid point

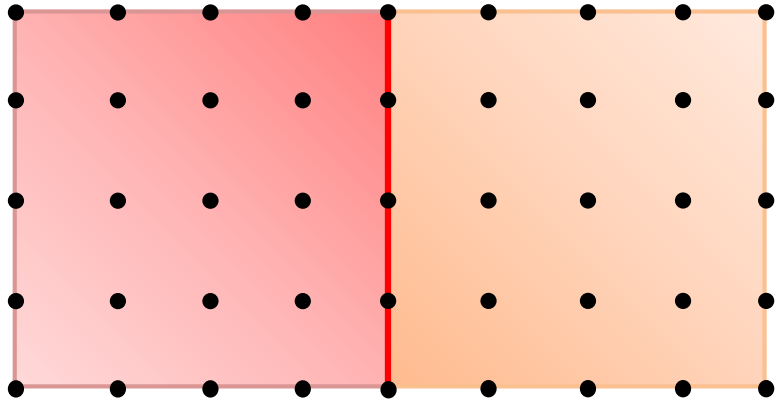


Figure 2.1: The interface between two cells shown from above.

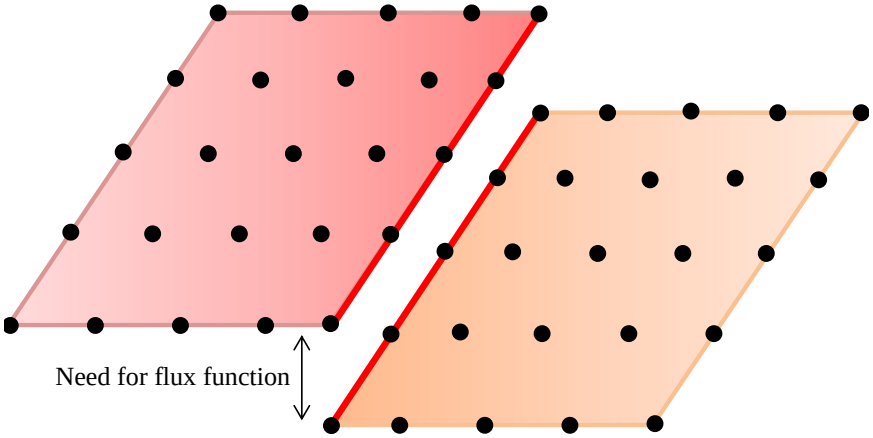


Figure 2.2: The interface between two cells shown from the side.

spacing is only representative, but the number of points shows what would be comparable to a fourth order polynomial for the cells. The red line denotes where the problem occurs at the interface, and the shading represents the magnitude of some value like density or velocity across the cell. From above, it is hard to appreciate where the problem arises, but the view from the side demonstrates that what appears to be one node location from above is actually two - each with a different value. Therefore, the question then becomes how to condense two values at a particular point to just one. There are infinite options for doing this, including a simple average of the two or else some weighting formula that counts the value from one side of the interface more heavily than the other. All of these are known as numerical flux

functions, although there are several preferred options that have proven stable and effective in the Finite Volume (FV) community, where discontinuity between cells is also an issue. Their main role is to provide stability by incorporating the physics and direction of the flow.

2.1 Choice of Numerical Flux Function

Since there are so many ways of going about reducing two values to one, there are some restrictions that must be placed to ensure stability and accuracy. In order for any given numerical flux function to qualify, a few basic criteria must be met: given two identical states, the flux function must return the original state, and the flux at an interface should be equal and opposite if only the order of states is swapped and the direction of the normal vector is reversed. The direction of the outward facing normals (which provide mathematical rules for what is considered the 'left' and 'right' states) is shown in Figure 2.3. With f^*

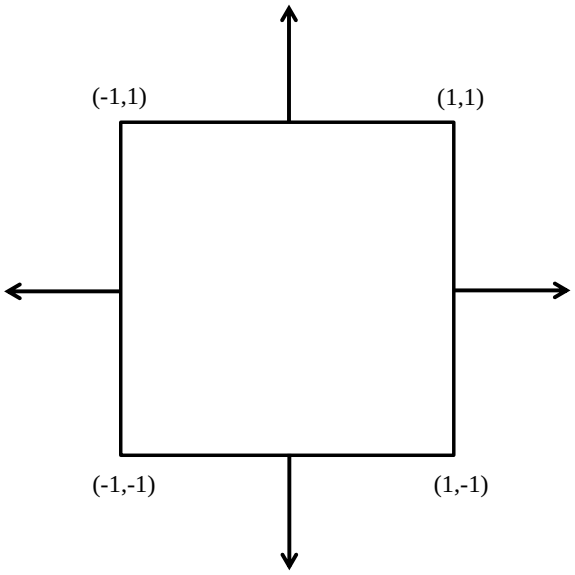


Figure 2.3: Outward facing normals on a standard [-1,1] cell

denoting a numerical flux and q^- and q^+ referring to the left and right sides of the interface

respectively, these conditions are expressed mathematically as

$$\begin{aligned} f^*(q, q\,;\, n_x) &\equiv f(q)\, n_x \\ f^*(q_h^{k,-}, q_h^{k,+}; n_x^{k,-}) &= -f^*(q_h^{k,+}, q_h^{k,-}; n_x^{k,+}) \end{aligned} \tag{2.9}$$

While there are admittedly many choices that satisfy these requirements, the particular choice *does* matter and is the key influencer of stability - an essential consideration in any computational scheme. Two particular schemes that have proven stable and effective are the Lax-Friedrichs and AUSM$^+$ -up flux functions, and their basic qualities are outlined below.

2.1.1 Lax-Friedrichs Flux Function

Given that a flux function takes two states (q_h^+ and q_h^-) on opposite sides of an interface as well as a normal vector (n_x) and produces a flux vector f^*, the Lax-Friedrichs (LF) flux is defined as:

$$f^* = \frac{f(q_h^+) + f(q_h^-)}{2} \;-\; |f'(q_h)|_{\max} \frac{q_h^+ - q_h^-}{2} n_x^- \tag{2.10}$$

Since the LF flux considers both q^+ and q^-, it is known as a central flux, which is to be distinguished from other fluxes that are perhaps upwind (those that consider information only from the direction it is coming). It is also significant to note that the 'stencil', or breadth of points needed to construct the flux, is relatively narrow in that only the directly neighboring information is needed. Other schemes can require information from two cells or more away, but that presents problems near boundaries, where it becomes unclear how best to construct data that is not physically there. The LF flux is a simple and straightforward option, but is not always the most accurate because of the nature of a central flux. Since they draw some information from downstream, information is included in the flux calculation which does not physically affect the point or interface.

2.1.2 AUSM$^+$-up Flux Function

Another option is the AUSM$^+$-up scheme, which is closely related to AUSM$^+$ (Advection Upstream Splitting Method), but the '-up' denotes that it has been modified to accommodate all Mach numbers - namely those approaching zero. Considering the flow does indeed approach $M = 0$ at the vortex centers and near the edges of the domain, this quality is important.

The AUSM$^+$ -up scheme is more complex than the LF flux, and begins with splitting the inviscid flux into convective and pressure fluxes such that

$$f^* = \dot{m}_{1/2} \begin{cases} \vec{\psi}_{\mathrm{L}} & \text{if } \dot{m}_{1/2} > 0, \\ \vec{\psi}_{\mathrm{R}} & \text{otherwise,} \end{cases} + \mathbf{p}_{1/2} \tag{2.11}$$

where in one dimension, $\vec{\psi}_{\mathrm{L/R}} = (1, u_{\mathrm{L/R}}, e_t + \frac{p}{\rho})^{\mathrm{T}}$ and

$$\dot{m}_{1/2} = a_{1/2} M_{1/2} \begin{cases} \rho_L & \text{if } M_{1/2} > 0, \\ \rho_R & \text{otherwise} \end{cases} \tag{2.12}$$

The subscripts L or R denote either the left or right state (to be consistent with the method in literature), and the "1/2" subscripts indicate the value is at the interface between the two cells. Unwinding is employed, as the left state is always used when $M_{1/2}$ is positive, or the 'wind is blowing' from left to right, and the right state is used when the 'wind' is blowing from right to left (or $M_{1/2}$ is negative). Since the scheme incorporates the physics of the flow by upwinding and is quite robust, it is a good choice and was selected to be used for all of the work herein. For more extensive information about the AUSM$^+$ -up flux, see work by Liou [18, 19].

Now that a numerical flux is being employed, Equation 2.8 reads

$$\int_{D^k} b_i^k \frac{\partial q_h^k}{\partial t} \,\mathrm{d}x + \oint_{\partial D^k} b_i^k \, f^*(q_h^{k,-}, q_h^{k,+}; n_x^{k,-}) \,\mathrm{d}s - \int_{D^k} \frac{\partial b_i^k}{\partial x} f_h^k \,\mathrm{d}x = 0 \tag{2.13}$$

with the understanding that f^* will be calculated in the AUSM$^+$-up fashion.

2.2 Basis Function Selection

Now that the problem with the middle term of Equation 2.8 has been resolved, it is time to consider a choice for the basis function. While many functions would work, polynomials are a logical choice for our familiarity with them. Still, even polynomials provide various options. Narrowing even further, we might consider a polynomial of either a modal or nodal variety for their specific properties to be discussed later. Recalling the earlier formulation of Equation 2.3, which had some coefficient multiplied by some basis function, the state can be expressed with a basis on a standard element ($\xi \in [-1, 1]$) as follows:

$$q_h(\xi) = \sum_{m=0}^{N} \tilde{q}_m \pi_m(\xi) = \sum_{n=0}^{N} \hat{q}_n \ell_n(\xi) \tag{2.14}$$

Here, the q's are still coefficients, but the basis b has been replaced by either π_m or ℓ_n to represent a modal or nodal basis, respectively. The two versions are numerically equivalent, but their respective formulation leads to pros and cons for each. Examples of each type of basis are plotted in Figures 2.4 and 2.5 up to fifth degree.

Based on the plots, the modal and nodal approaches are clearly quite different from each other, so it is important to assess the benefits and drawbacks that each may present before choosing one. In taking a closer look at the modal basis, it turns out that the coefficients $\tilde{q}_m$ are found through a matrix inversion. Matrix inversions are notoriously problematic, as they can be quite difficult (and time consuming) to perform, and more importantly, to perform accurately. The accuracy of inversion is measured by the condition number, $\kappa(A)$, and for some choices within the modal basis family, the condition number grows very quickly with increasing polynomial order (the condition number should be close to 1 for a well-conditioned, and thus accurately invertible, matrix).

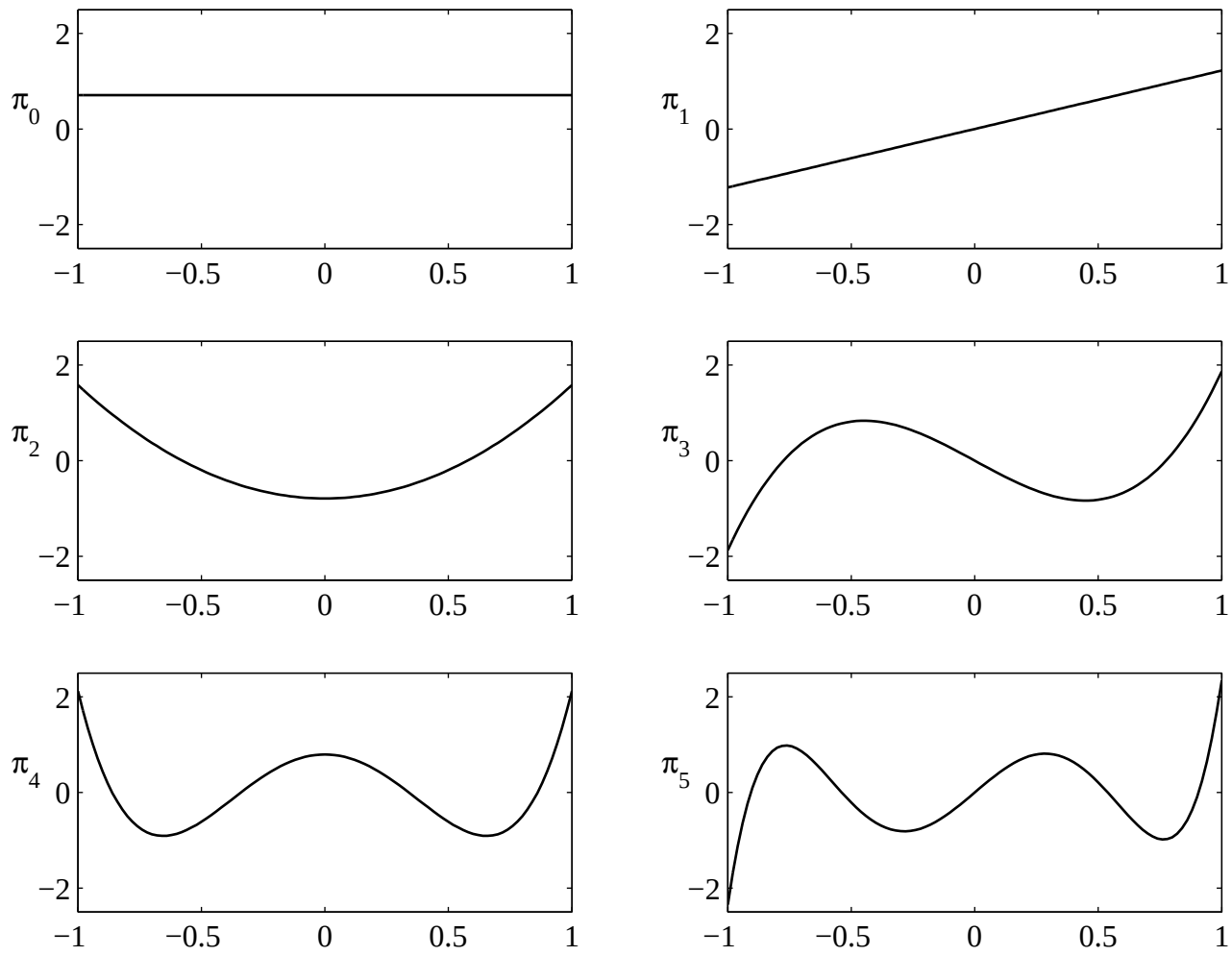


Figure 2.4: Legendre polynomials in $\mathbb{P}_5$

A good option within the modal basis realm that *does not* lead to a poorly conditioned matrix involves using an orthonormal basis. Again, there are several variations, but Legendre polynomials are a well-known type and have an important property due to their orthonormality. Namely:

$$\int_{-1}^{1} \pi_i(\xi)\pi_j(\xi)\mathrm{d}\xi = \delta_{ij} \doteq \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{2.15}$$

In other words, the integrated product of two of the same polynomials over a standard cell will be 1 and the integrated product of two different polynomials will be 0. This is obviously

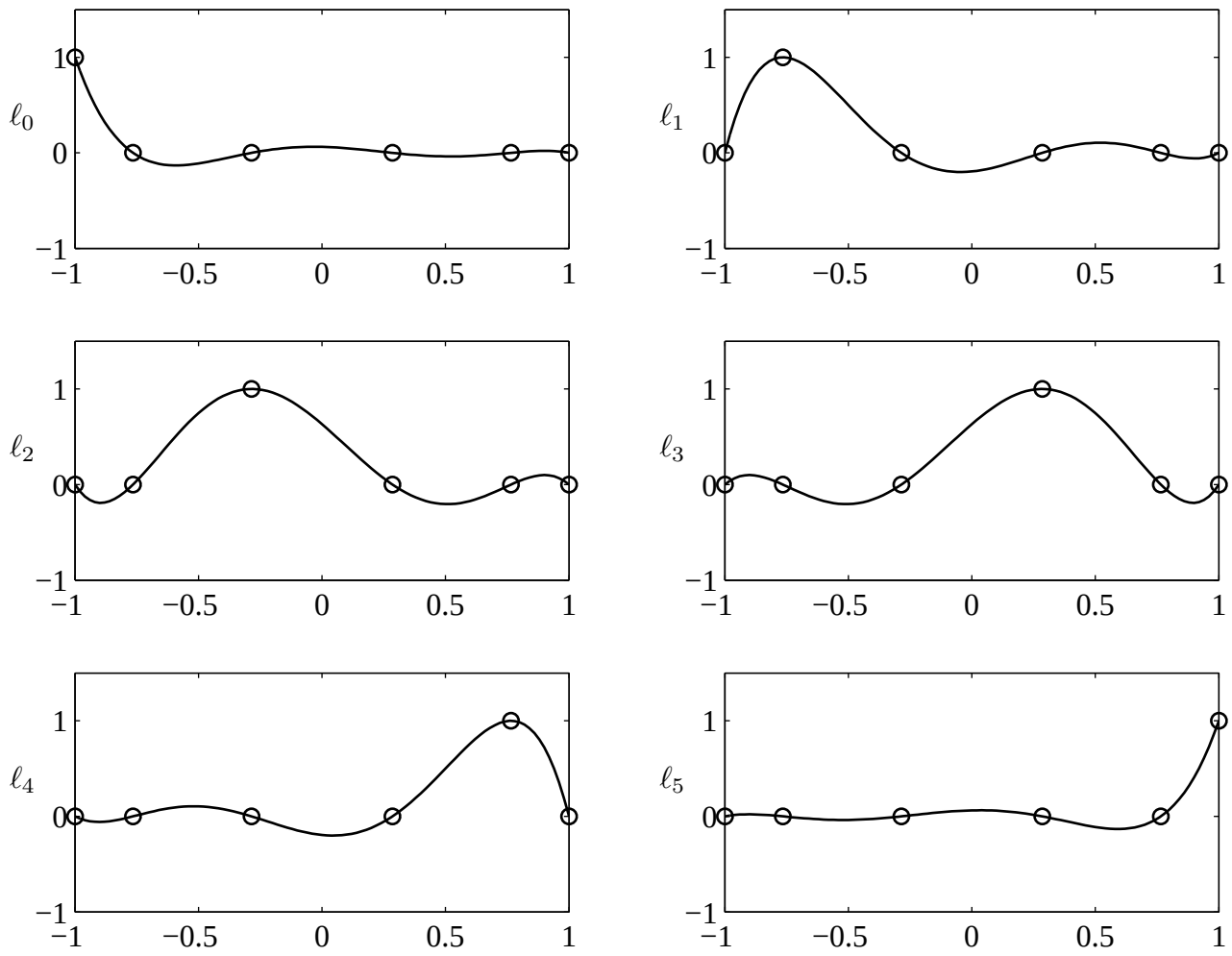


Figure 2.5: Lagrange polynomials in $\mathbb{P}_5$

a desirable property if the product of two bases arises, as the answer is straightforward and could lead to cancellation of terms altogether if $i \neq j$.

Moving on to the nodal approach, the most notable difference is that all of the polynomials are of degree N (they don't build sequentially from 0 up to N as in the modal Legendre polynomials). The nodal approach works like it sounds, as it involves interpolating between certain points, or nodes. Lagrange polynomials are a well-known interpolating polynomial, and are plotted in Figure 2.5. The open circles depict the points $\xi_0, \xi_1, ..., \xi_N$ to

be interpolated, and the polynomials are described by

$$\ell_n(\xi) = \prod_{\substack{i=0 \\ i \neq n}}^{N} \frac{\xi - \xi_i}{\xi_n - \xi_i} = \left(\frac{\xi - \xi_0}{\xi_n - \xi_0}\right) ... \left(\frac{\xi - \xi_{n-1}}{\xi_n - \xi_{n-1}}\right) \left(\frac{\xi - \xi_{n+1}}{\xi_n - \xi_{n+1}}\right) ... \left(\frac{\xi - \xi_N}{\xi_n - \xi_N}\right) \tag{2.16}$$

Lagrange polynomials also possess a useful property, which is that the polynomial evaluates to zero at every point but one (where it happens to equal 1). Since the polynomial is only valued 1 once, the coefficients $\hat{q}_n$ in Equation 2.14 are easily found and are simply the function values at those points - no matrix inversion necessary. This is expressed mathematically as

$$\ell_j(\xi_i) = \delta_{ij} \doteq \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{2.17}$$

and put even more simply, means that $\ell_j(\xi_j) = 1$.

Although the Legendre and Lagrange polynomials seem very different, they are in fact related via a simple transformation. If the Legendre polynomials are evaluated at the same points as the Lagrange polynomials, the result is none other than the Lagrange polynomials themselves. The matrix used to achieve this transformation is called the Vandermonde matrix (V). Using the Vandermonde matrix means that going from modal (denoted by tildes) to nodal (denoted by hats) is quite easy, and the converse is also true as long as V is accurately invertible.

$$V\tilde{q} = \hat{q} \quad \Longleftrightarrow \quad \tilde{q} = V^{-1}\hat{q} \tag{2.18}$$

$$\text{modal} \rightarrow \text{nodal} \qquad \text{modal} \leftarrow \text{nodal}$$

Again, the invertibility of V raises some concerns, and since V is created by evaluating the Legendre polynomials at the same points as the Lagrange polynomials, it makes sense that the choice of where to place those N points for the Lagrange polynomials could affect the invertibility of V. This is indeed true, and a special relation defines the Gauss-Lobatto (GL) quadrature points which have several unique properties, the first of which is that they ensure

the best possible conditioning for V. Furthermore, it should be noted that "quadrature" is the approximation of an integral and involves a weighted sum at certain points so that the integral of some function f is approximated as

$$\int_{-1}^{1} f(\xi)\mathrm{d}\xi \simeq \sum_{j=1}^{N_p} \omega_j f(\xi_j) \tag{2.19}$$

where ω_j are the quadrature weights, and ξ_j are the quadrature nodes (the specific points at which the weights will be applied to get the overall integral). It turns out that for polynomials at least, equally spaced points are not the best option and instead, the slightly offset Gauss-Lobatto quadrature points are a better choice and actually yield *exact* results. For a look at some sample weights and nodal locations in a cell, see Figures 2.6 and 2.7. Notice that the nodes are displaced more toward the edges of the cell compared to their locations if they were equally spaced.

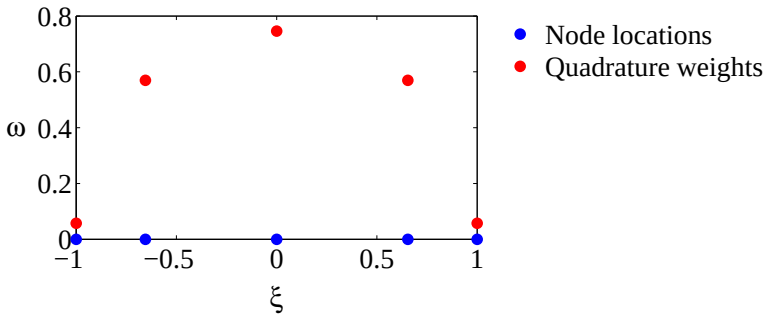


Figure 2.6: The Gauss-Lobatto nodal locations ξ with their associated weights ω for a 4^{th} degree polynomial

In conclusion, a nodal basis makes the most sense because its pointwise properties make getting boundary data straightforward (no matrix inversion is needed) and handles the discontinuous flux issues of Section 2.1 with relative ease. Even though it means losing the orthonormality of the modal basis, those benefits can be accessed by using the Vandermonde matrix to switch back and forth depending on the situation. Additionally, the use of the Gauss-Lobatto points guarantees accuracy both in any inversions of V and in integration.

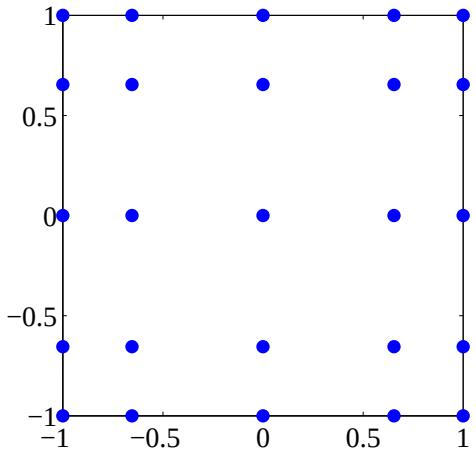


Figure 2.7: A sample 2D cell with Gauss-Lobatto points for a 4^{th} degree polynomial

2.3 Putting it Together

Now that the general theory has been outlined and an appropriate numerical flux and basis have been chosen, the results from the preceding sections can be collected to further develop the DG method and prepare it for use in simulation. To begin, since the standard computational cell on the $[-1, 1]$ interval is generally a different size than the physical cell, a transformation is needed when converting between the two. If the node locations on the standard cell are ξ_i and correspond to the physical node locations x_i, those locations can be found with

$$x_i = x_a + \frac{\xi_i + 1}{2}(x_b - x_a) \tag{2.20}$$

where x_a and x_b are the left and right edges of the cell, respectively.

Another transformation is needed when using differentials, as the cell side-length is also different for the computational and physical cells. While the side length for the computational cell is always 2, the side length of the physical cell will vary depending on the number of cells chosen to represent the domain (more cells will result in a smaller side length). To accurately account for this, the ratio of the lengths must be considered, and the result is the Jacobian, $\mathcal{J}$:

$$\mathrm{d}x^k = \mathcal{J}^k \, \mathrm{d}\xi, \quad \mathcal{J}^k = \frac{x_b^k - x_a^k}{2} \tag{2.21}$$

If all the cells are the Cartesian and the same, the values calculated with Equation 2.21 will be identical for all cells, but if an adaptive or unstructured grid is employed, $\mathrm{d}x^k$ and $\mathcal{J}^k$ will need to be calculated for each cell individually. However, since all grids used for this work were uniform structured Cartesian grids, the k will be dropped and the Jacobian will be denoted as simply $\mathcal{J}$, as it remains constant for all cells in the domain (along with Δx).

To incorporate the conclusions of Sections 2.1 and 2.2 and develop some useful operators, we now revisit Equation 2.13:

$$\underbrace{\int_{D^k} b_i^k \frac{\partial q_h^k}{\partial t}\,\mathrm{d}x}_{①} + \underbrace{\oint_{\partial D^k} b_i^k\, f^*(q_h^{k,-}, q_h^{k,+}; n_x^{k,-})\,\mathrm{d}s}_{②} - \underbrace{\int_{D^k} \frac{\partial b_i^k}{\partial x} f_h^k\,\mathrm{d}x}_{③} = 0$$

From here forward, b_i will be replaced with ℓ_i to denote the choice of Legendre polynomials as the basis function. In the following derivations, it is important to remember that *any* function q_h can be expressed with a basis so that $q_h(\xi) = \sum_{i=0}^{N} \hat{q}_i \ell_i(\xi)$. Looking at term ①, the time derivative term can be decomposed with a basis by letting $\frac{\partial q_h^k}{\partial t} = \frac{\partial}{\partial t}\left(\sum_{j=0}^{N} \hat{q}_j^k(t)\ell_j^k(x)\right)$, and after some rearranging, the term can be simplified to

$$\int_{D^k} \ell_i^k \frac{\partial q_h^k}{\partial t}\,\mathrm{d}x = \mathcal{J}\sum_{j=0}^{N} \mathcal{M}_{ij}\frac{\mathrm{d}\tilde{q}_j^k}{\mathrm{d}t} \tag{2.22}$$

where $\mathcal{M}_{ij}$ is the mass matrix and is defined

$$\mathcal{M}_{ij} \doteq \int_{-1}^{1} \ell_i^k(\xi)\ell_j^k(\xi)\mathrm{d}\xi \tag{2.23}$$

Much like with term ①, the numerical flux in term ② can also be written with a basis function and simplified to

$$\oint_{\partial D^k} \ell_i^k\, f^*(q_h^{k,-}, q_h^{k,+}; n_x^{k,-})\,\mathrm{d}s = \sum_{m=0}^{N}(L_a)_{im}(\tilde{f}_a^*)_m^k + \sum_{n=0}^{N}(L_b)_{in}(\tilde{f}_b^*)_n^k \tag{2.24}$$

where L_a and L_b are the lifting matrices:

$$(L_a)_{ij} \doteq \ell_i(-1)\ell_m(-1), \quad (L_b)_{ij} \doteq \ell_i(1)\ell_n(1) \tag{2.25}$$

They are so named because they "lift" the boundary data by one dimension - for example, 1D to 2D. They involve the basis evaluated at the boundaries because the numerical flux does not exist inside the element.

Finally, term ③ can similarly be rewritten using a basis to yield

$$\int_{D^k} \frac{\partial \ell_i^k}{\partial x} f_h^k \, \mathrm{d}x = \sum_{j=0}^{N} (S^T)_{ij} \tilde{f}_j^k \tag{2.26}$$

where S is the stiffness matrix and is defined

$$\mathcal{S}_{ij} \doteq \int_{-1}^{1} \ell_i(\xi) \frac{\mathrm{d}\ell_j(\xi)}{\mathrm{d}\xi} \mathrm{d}\xi \tag{2.27}$$

Here, it should be pointed out that $\tilde{f}_j^k$ is the cell flux (not to be confused with the numerical flux). In one dimension, each node has a state value of $q = (\rho, \rho u, \rho e_t)$ that can be used to construct an associated flux vector where $\tilde{f} = (\rho u, \rho u^2, (\rho e_t + p)u)$ and k has been omitted for clarity. For instance, this means that a cell with N_p nodes would have a cell flux matrix that is N_p x 3.

Combining the results from terms ①, ②, and ③, Equation 2.13 can now be expressed in matrix-vector form as

$$\mathcal{J}\mathcal{M} \frac{\mathrm{d}\tilde{q}^k}{\mathrm{d}t} + L_a(\tilde{f}_a^*)^k + L_b(\tilde{f}_b^*)^k - \mathcal{S}\tilde{f}^k = 0 \tag{2.28}$$

Solving for $\dfrac{\mathrm{d}q^k}{\mathrm{d}t}$ results in the equation needed to propagate the solution in time.

$$\frac{\mathrm{d}\tilde{q}^k}{\mathrm{d}t} = \mathcal{J}^{-1}\mathcal{M}^{-1} \left[S^T \tilde{f}^k - L_a(\tilde{f}_a^*)^k - L_b(\tilde{f}_b^*)^k \right] \tag{2.29}$$

The result is valid for each cell k and can be implemented one cell at a time to construct the solution over the entire domain for each time step. This ability to solve the equation one cell at a time is much faster and more efficient than solving the whole domain at once and is one of the biggest benefits of the DG method.

2.4 Implementation

Equation 2.29 is a big step toward implementing the DG method, but it still isn't clear exactly how to create the operators $\mathcal{M}$, S,and L. However, using the fundamental relationship that $\ell_i(\xi) = \sum_{j=0}^{N}(V^{-T})_{ij}\pi_j(\xi)$, they can be derived as follows and implemented in MATLAB or Fortran.

$$\mathcal{M}^{-1} = VV^T \tag{2.30}$$

$$S^T = \left(V_\xi V^{-1}\right)^T (VV^T)^{-1} \tag{2.31}$$

$$L_a = \vec{e}_0, \quad L_b = \vec{e}_N \tag{2.32}$$

$$V_{ij} \doteq \pi_j(\xi_j), \quad (V_\xi)_{ij} \doteq \pi_j'(\xi_i) \tag{2.33}$$

2.4.1 In Two Dimensions

All theory discussed so far has been in one dimension for simplicity's sake. Fortunately, extension of the DG method from one dimension to two is relatively straightforward. The first major modification is that the state changes from $q = (\rho, \rho u, \rho e_t)$ to $q = (\rho, \rho u, \rho v, \rho e_t)$. The other changes are mainly composed of expanding all of the operators by a factor of N_p in each direction and adding a y-component of each operator. For instance, S is replaced by S_x and S_y and goes from N_p x N_p to S_x and S_y being N_p^2 x N_p^2 each, and all other operators like V and $\mathcal{M}$ are similarly altered.

The lifting matrices are also subdivided into L_x and L_y, but are a little trickier because the cell edge changes so much. In one dimension, the edges of a cell are one point each, with the lifting matrix for the left boundary being $\vec{e}_0$ and the lifting matrix for the right

boundary being $\vec{e}_N$. However, in two dimensions, the boundary is no longer composed of just one point and is rather a *set* of points - the specific ones depend on which edge of the cell (left/right for L_x and top/bottom for L_y). Figure 2.8 shows how the nodes in a cell are numbered and which ones are used for each lifting matrix so that the structure of the matrices in Figure 2.9 is more easily understood. It can be seen that the new size for the lifting matrix in two dimensions is now N_p^2 x N_p instead of N_p x 1. The blue markers show where there are nonzero entries, and the rows that are nonzero correspond to the number of the node located on the cell edge. As before, a and b denote whether the lifting is applied to the left (a) or right (b) side of the cell.

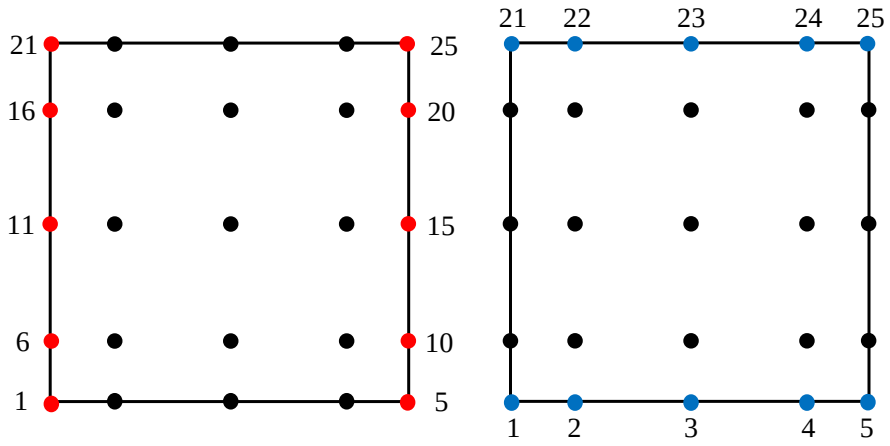


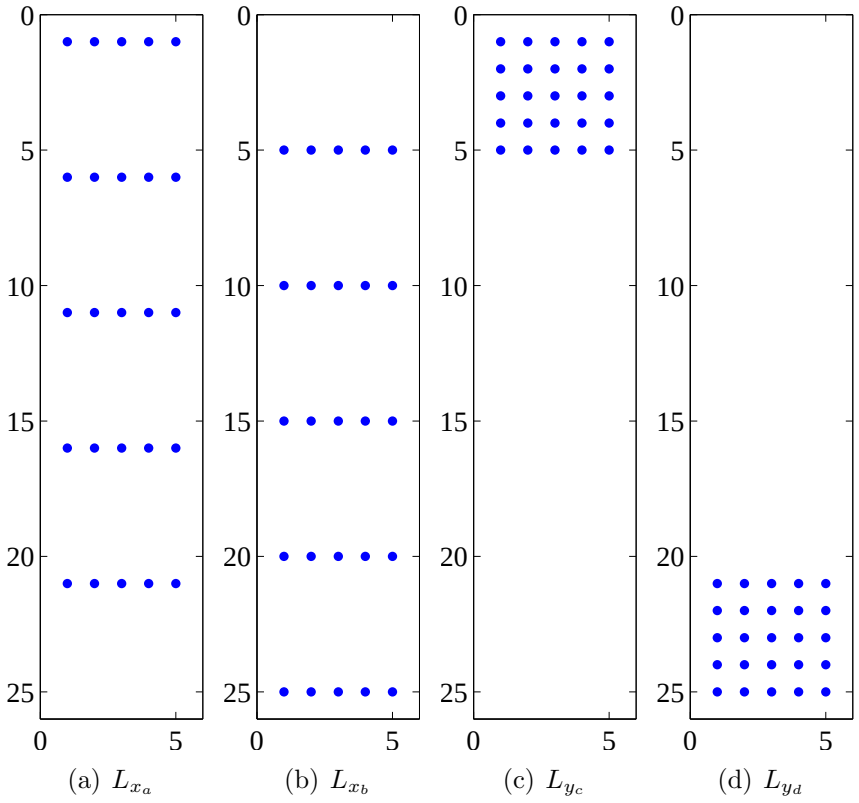
Figure 2.8: Boundary nodes. Red are used for L_x and blue are used for L_y

.

To recap, for two dimensions, equation 2.29 now looks like:

$$\frac{\mathrm{d}\tilde{q}^k}{\mathrm{d}t} = (\mathcal{J}^k)^{-1}\mathcal{M}^{-1}\left[S_x^T\tilde{f}^k + S_y^T\tilde{g}^k - L_{x_a}(\tilde{f}_a^*)^k - L_{x_b}(\tilde{f}_b^*)^k - L_{y_c}(\tilde{f}_c^*) - L_{y_d}(\tilde{f}_d^*)\right] \tag{2.34}$$

The c and d subscripts on the lifting matrices and numerical fluxes indicate that they are for the top and bottom boundaries of the cells, and the a and b still refer to the left and right sides of the cell. The cell flux, $\tilde{f}$, is now comprised of four terms, such that $\tilde{f} = [\,\rho u,\, \rho u^2{+}p,\, \rho uv,\, (\rho e_t{+}p)u\,]$, and there is a y-counterpart, $\tilde{g} = [\,\rho v,\, \rho uv,\, \rho v^2{+}p,\, (\rho e_t{+}p)v\,]$.



(a) L_{x_a} (b) L_{x_b} (c) L_{y_c} (d) L_{y_d}

Figure 2.9: Sample 2D lifting matrices for a 4^{th} degree polynomial

Equation 2.34 now fully describes the basic DG method as it is used in this work to model two-dimensional flows.

2.4.2 Advancing in Time

Once the state is constructed for the entire domain, the only remaining question is how to advance the solution in time. The first consideration is what size time step to use, and it is restricted by the CFL condition - an expression that includes cell size and polynomial degree. The result is an equation for Δt that limits how large the time step can be:

$$\Delta t = \frac{C\Delta x}{\lambda_{\max} d^2} \tag{2.35}$$

Here, $\lambda_{\max}$ is the maximum wave speed and C is the CFL number. As one can see, either increasing the degree of the polynomial or decreasing the size of the cells (in other words,

increasing cell density) will both result in a smaller time step. Therefore, the smallest time step is required for those cases with *both* a high cell density and high degree.

With the time step decided and $\mathrm{d}q/\mathrm{d}t$ calculated, it only remains to determine how $\mathrm{d}q/\mathrm{d}t$ should be used to update the state. In this work, a 3-stage Runge-Kutta time-update was used with the following coefficients, where the first column contains the coefficients for stage 1, and so on.

$$\alpha_{i,s} = \begin{bmatrix} 1 & \frac{3}{4} & \frac{1}{3} \\ 0 & \frac{1}{4} & \frac{2}{3} \\ 1 & \frac{1}{4} & \frac{2}{3} \end{bmatrix}$$

Chapter 3

Lagrangian Vortex Simulation

Now that the groundwork has been laid for the DG method, it can be applied to a test problem. As outlined in Chapter 1, the ultimate goal is to better understand three dimensional vortex interaction, but less complex cases must first be considered in order to discover some of the nuances of choosing parameters for the DG method. Two-dimensional counter-rotating vortices were chosen as a test problem for their simplicity and well-documented behavior.

Before implementing the DG method to obtain any CFD results, however, another method is needed for comparison. To that end, a four-vortex system is modeled with a Lagrangian approach. In contrast to CFD, where the entire domain is subdivided and modeled with elements, a Lagrangian method treats a vortex as a point that induces a velocity field felt by the other vortices.

The basic idea of the method is depicted in Figure 3.1, which shows vortex j located at (X_j, Y_j) and the velocity v it induces on vortex i. The induced velocity causes vortex i to move, and that motion is described by

$$\frac{\mathrm{d}}{\mathrm{d}t}X_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} \frac{\Gamma_j}{2\pi r_{ij}^2}(-\Delta y_{ij}) \tag{3.1a}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}Y_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} \frac{\Gamma_j}{2\pi r_{ij}^2}(+\Delta x_{ij}) \tag{3.1b}$$

where $\vec{r}_{ij}$ is the distance between the two vortices, Γ_j is the circulation of vortex j, and Δx_{ij} and Δy_{ij} are the x- and y- components of $\vec{r}_{ij}$, respectively. The summation ensures that the effect of all vortices except itself (i in this case) are included so that the motion is due to all

vortices nearby. In Figure 3.1, there is only one other vortex, but Equation 3.1 also applies to an array of vortices.

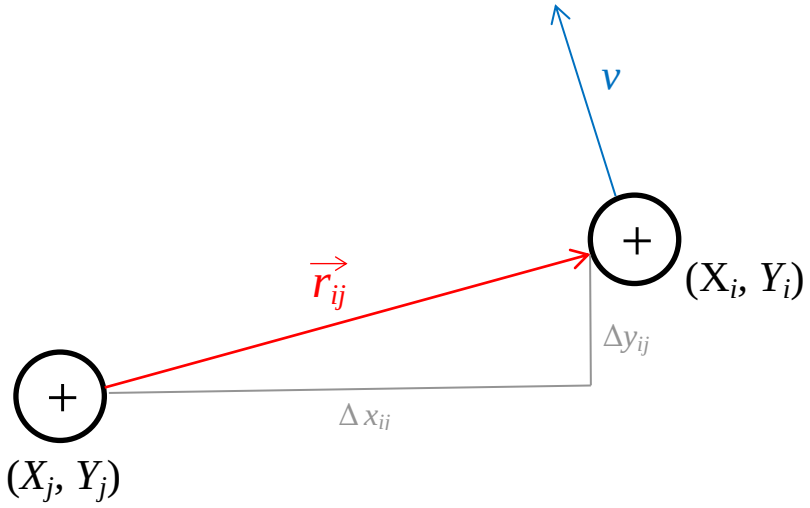


Figure 3.1: Induced velocity from vortex j on vortex i

3.1 Vortex Selection

The first step is to choose a vortex profile to use, as there are many options, including high order algebraic, Lamb-Oseen, isentropic/Taylor, and so on. For this work, a Gaussian profile was selected for the realistic properties and simple formulation. The size of the vortex is easily controlled by setting the radius where the maximum velocity occurs and is denoted by r_c. Since all lengths used in this work were nondimensionalized by r_c, it was set to 1 (after being normalized by itself).

Regardless of the vortex type used, the initial state is built by adding perturbations to the free stream flow, where the circulation profile $\Gamma(r)$ determines the shape and characteristics of the vortex produced. The tangential velocity profile is related through $u_\theta(r) =$

$\Gamma(r)/2\pi r$. The perturbations to the velocities u and v are then constructed as follows:

$$\Delta u = -f\Delta y \tag{3.2a}$$

$$\Delta v = f\Delta x \tag{3.2b}$$

where the variable f is a convenient quantity used to represent $u_\theta(r)/r$. The specific form of f for Gaussian vortices is described by

$$f = \frac{\Gamma_0}{2\pi r^2}\left(1 - e^{-1.256\left(\frac{r}{r_c}\right)^2}\right) \tag{3.3}$$

where Γ_0 is the total circulation, r_c is the radius of the vortex, and r is the distance between a point (x, y) and the center (x_0, y_0) of the vortex, so that $r = ((x - x_0)^2 + (y - y_0)^2)^{1/2}$ [20]. Figure 3.2 shows a comparison between Taylor (isentropic) and Gaussian vortices for approximately the same maximum velocity. The inner profile is similar, but beyond the maximum velocity (denoted by the red line positioned at r_c), the profile is much different and the Gaussian vortex's influence extends much farther. This is more consistent with vortices observed experimentally and with those found in classic theory.

3.2 System Setup

The general idea of Section 3.1 is then extended to a system of *four* vortices, with the intention that they represent the two wing and two tail vortices of a fixed-wing aircraft. Rather than choose arbitrary numbers, Boeing 747 information is used to set vortex radii, circulation, and wingspan values, since it is exactly the type of large transport aircraft that creates strong wakes. The 747 has a wingspan of 64.4 meters, so once normalized by its 2.6 meter wing vortex radius, the centers of the vortices are located at [-12.385, -3.715, 3.715, 12.385].

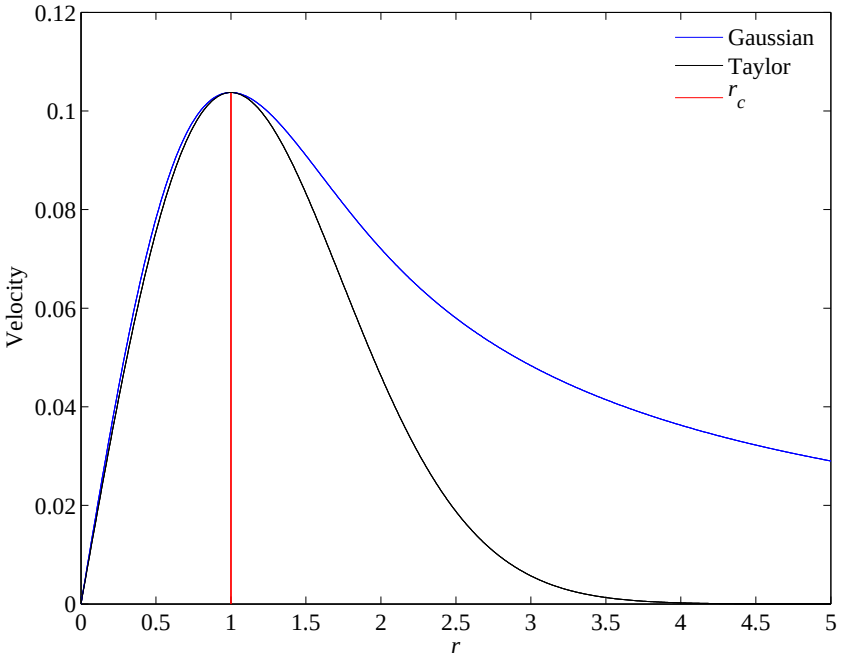


Figure 3.2: Comparison of velocity profiles

For a visualization of the four-vortex system, see Figures 3.3 and 3.4. The red vortices correspond to the wing and the blue vortices are from the tail. Taken together, they are two sets of counter-rotating vortices - one set on each side of the aircraft. The particular circulation values for the vortices come about because total circulation Γ_0 for the system is defined so that lift balances weight. Then, the tail is used to trim the airplane by producing downward force which also creates vortices of opposite sign and lower strength than the wing vortices. If Γ_1 corresponds to the wing vortices and Γ_2 to the tail vortices, the relationship is as follows:

$$\Gamma_0 = \Gamma_1 + \Gamma_2 \tag{3.4a}$$

$$\Gamma_0 = .7\,\Gamma_1 \tag{3.4b}$$

$$\Gamma_2 = -.3\,\Gamma_1 \tag{3.4c}$$

A ratio of -.3 between Γ_1 and Γ_2 was selected because of findings in the FAR-Wake program that showed it to be optimal [21].

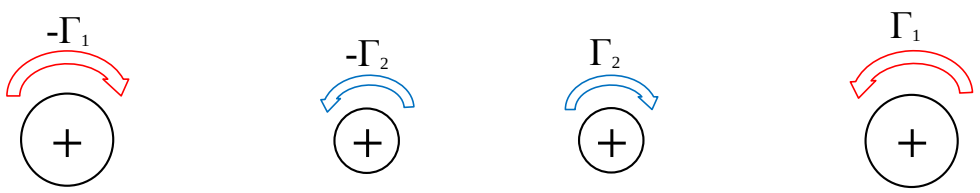


Figure 3.3: 4 vortex setup

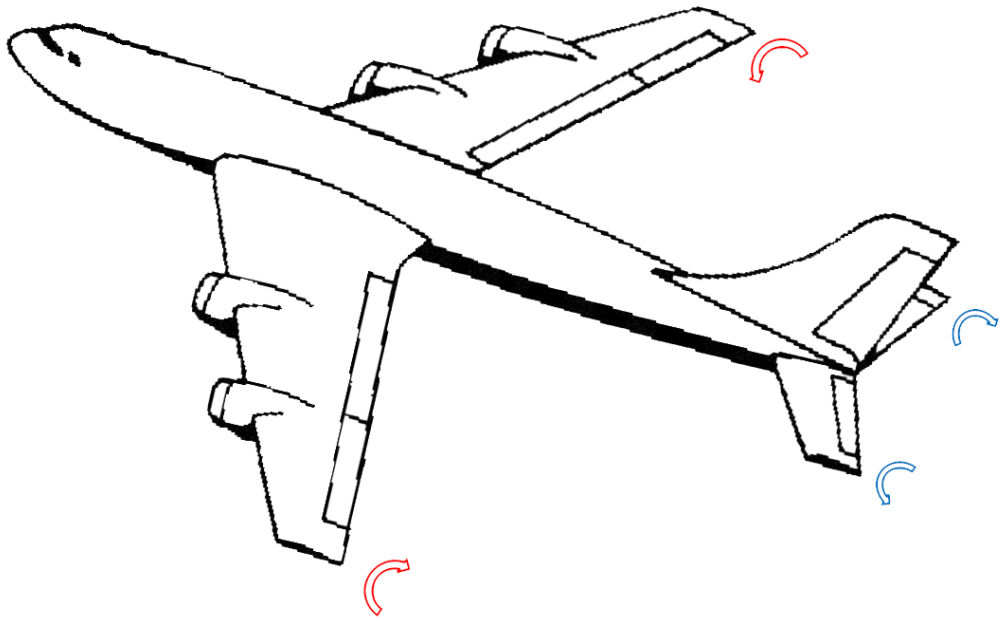


Figure 3.4: Diagram of vortices to be modeled

3.3 Boundary Conditions

Due to the nature of Lagrangian simulations and the fact that each *vortex* is visited instead of a node in a grid, the domain size becomes irrelevant. That is to say, the vortices can be modeled as if there are no boundaries at all. Trajectories of the vortex centers for such a case are plotted in Figure 3.5, where the tail vortex can be seen to rotate about the wing vortex and the pair convect in a general downward direction.

However, the Lagrangian results are meant to serve as an 'exact' result for comparison to CFD results, and CFD results can *not* be obtained without first defining where the boundaries should be located and how they should be treated. Thus, the Lagrangian approach

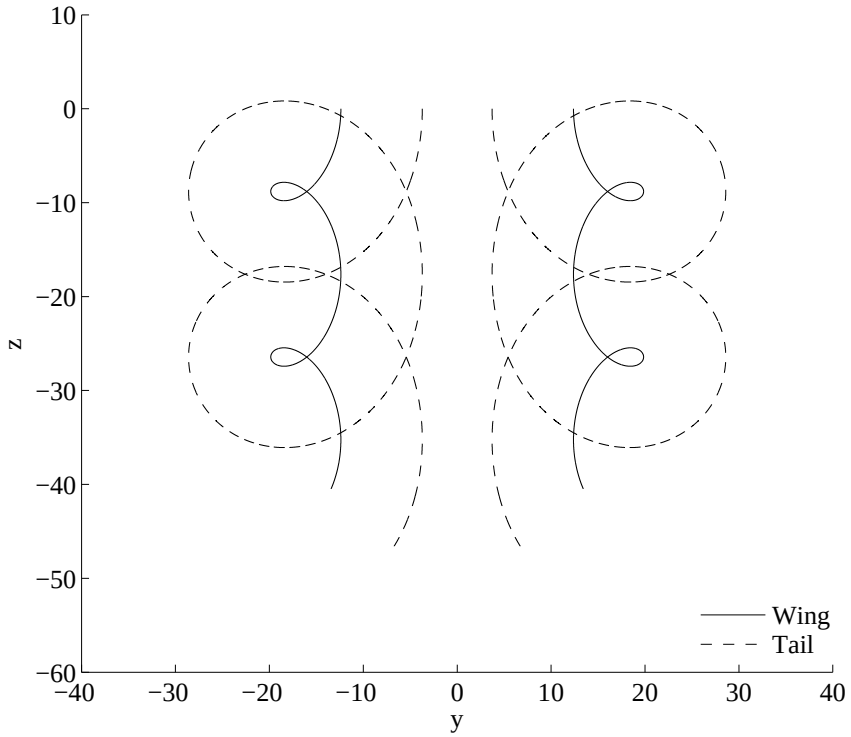


Figure 3.5: Lagrangian simulation of unbounded vortex system

can give a good idea of how unbounded vortices behave, but restricting them in a similar fashion as CFD also provides information about how bounded vortex systems behave, which can then be used for comparison with CFD results.

Unfortunately, confining the vortices is no simple task, as boundary conditions (and how to come up with them) are one of the most contested aspects of CFD. Should they be periodic or reflective, and *where* should they be defined? An ideal CFD simulation would place nonreflective boundaries so far away that their effect is negligible, but in reality, this is so complex and expensive that it is impractical. A more practical and common approach is to place the boundaries much closer and make them reflective. After all, almost all experimental results are obtained in wind or water tunnels with reflective walls, and even experiments conducted outside a laboratory are still subject to influence from the ground. With these reasons in mind, this work places the vortices in a 'box' with reflective boundaries much like a wind tunnel, as it seems more realistic than other options.

Once decided, the process of actually placing the boundaries and enforcing the reflective condition is relatively straightforward, but arriving at an initial condition that is consistent with the boundary conditions is not. Reflective boundary conditions are essentially the same as removing the boundaries and placing vortices of opposite sign as if reflected across the boundary. The process is continued until the original domain is surrounded by concentric 'rings' of vortices with alternating signs. Figure 3.6 depicts the first of these rings, where red outlines the actual domain boundary and orange outlines the first 'ring' of imaginary vortices, with the understanding that there are more rings in the same pattern that are not pictured. Since the induced velocity from a vortex is proportional to distance squared, the effects of the vortices in the more distant rings is minimal and can eventually be excluded as inconsequential.

It is important to note that the dashed vortices in Figure 3.6 don't physically exist, but their effects *do* because of the nature of reflective boundary walls. They are simply an illustration to help understand the implications of having reflective boundary conditions and how the implied vortices impact the four main vortices being modeled. The problem arises when the effects from the reflected vortices are not incorporated into the initial condition. If assumptions are made so they are not (as is often the case in literature), the result is an initial condition that does not match the boundary conditions. This work seeks to rectify that by including the effects of the reflected vortices in the initial condition to get the most accurate starting point possible.

To do that, the effects of 42 rings of vortices were sequentially added to the four main ones. While the distance of the furthest rings probably rendered their contribution negligible, it ensures complete convergence. Each of the four main vortices is visited so the effect of all the other vortices in the array (those from the 0^{th} through the 42^{nd} ring) on it can be found. The x and y velocity contributions are then stored in matrices u and v that are n x 4, where each row corresponds to one of the n rings and each column corresponds to one of the four main vortices. Thus, $u(11,3)$ is the x-velocity contribution from the 11^{th} ring on the third

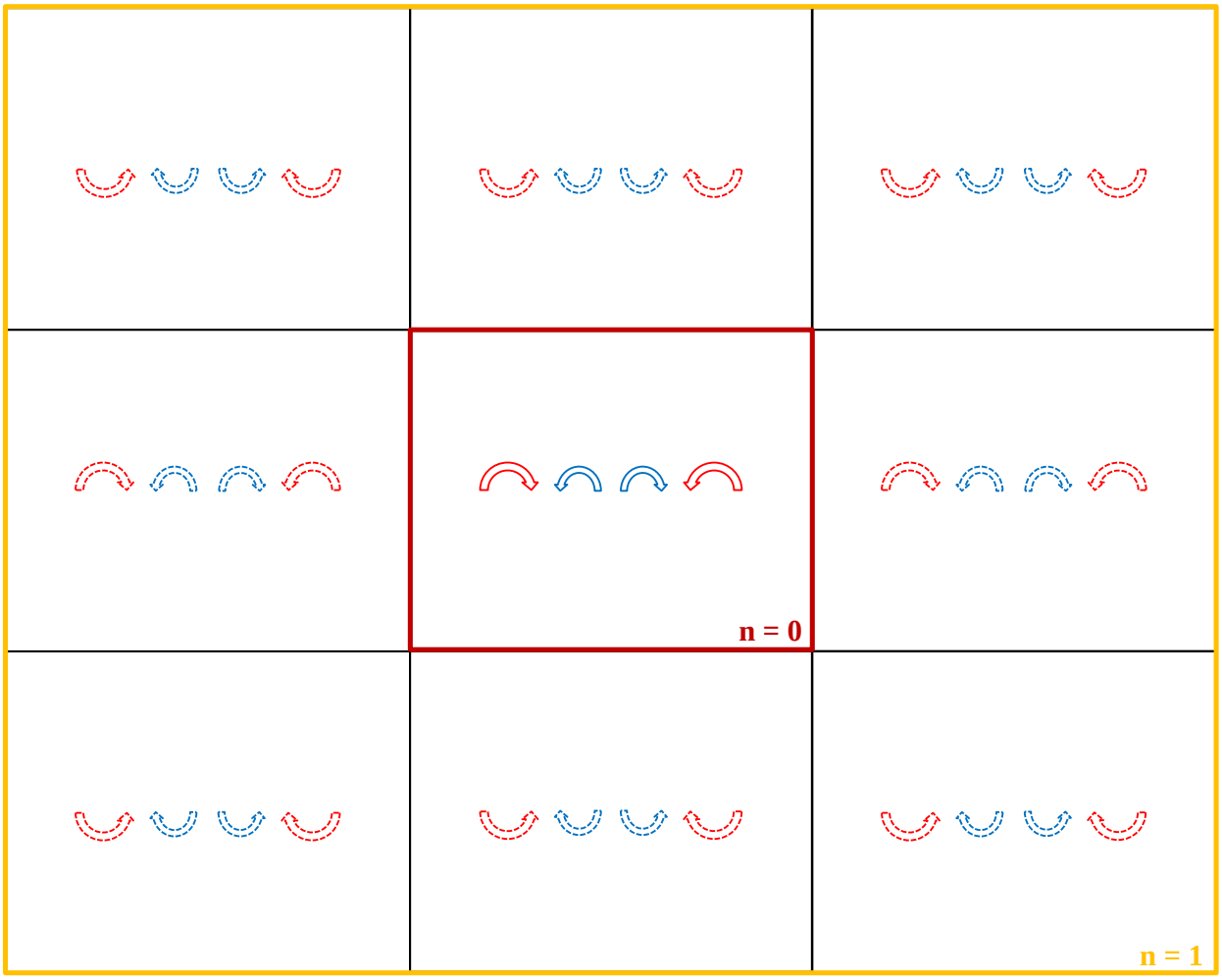


Figure 3.6: Vortices effectively created by the reflective boundaries

vortex. The matrices u and v are constructed with the following recursion:

$$u(n,i) = u(n,i) - si * sj * dy * f \tag{3.5a}$$

$$v(n,i) = v(n,i) + si * sj * dx * f \tag{3.5b}$$

The distances dx and dy are calculated from the i^{th} vortex (one of the main four) to any given 'contributor' vortex, which can be any vortex in the array but itself. The rotation direction for the contributor vortex is then determined by the product $si * sj$, where si and sj alternate between -1 and +1, depending on the contributing vortex's location. For

instance, a vortex that has only been reflected once in the x-direction would have values of $si = -1$ and $sj = +1$, but a vortex that has been reflected in both the x and y directions would have values of $si = -1$ and $sj = -1$. Essentially, it means that a vortex that has been reflected an odd number of times will result in $si * sj = -1$ and an even number of reflections will produce $si * sj = +1$. It is a complicated way of determining if the contributor vortex rotates in the same direction as the main vortex, as -1 means it does not and +1 means that it does (which conveniently reproduces Equation 3.2). It should also be noted that the process builds, so that if a contribution has already been calculated for a particular ring, the new contribution is added to it. This is why any row in u or v includes the effects of all the vortices in that ring, and not just one vortex.

3.4 Van Wijngaarden Acceleration

While the matrices u and v give each ring's effect on the vortices, they still don't provide the total effect of *all* the rings on the vortices. As the rings move further and further out, they contribute less and less to the velocity field, so the total effect of all the rings can be thought of as a series slowly converging to a value. Because of the form of Equation 3.5, it can be recognized specifically as an alternating series, and there are methods to accelerate its convergence.

Unsurprisingly, using more terms (rings in this case) ensures better accuracy, which is why so many rings are employed. The first step in converging a series s with K terms is to compute partial sums:

$$s_{0,k} = \sum_{n=0}^{k} (-1)^n a_n = a_0 - a_1 + a_2 - a_3 + ...a_k \quad \text{for } k = 0, 1, 2, ..., K \tag{3.6}$$

Each $s_{0,k}$ builds on the previous, so that $s_{0,4} = s_{0,3} + a_4$ and $s_{0,K}$ includes all available a_n terms. Then, adjacent terms from the 0^{th} row are averaged to produce $s_{1,k}$ and so on,

according to:

$$s_{j+1,k} = \frac{s_{j,k} + s_{j,k+1}}{2} \tag{3.7}$$

The averaging normally continues until the single value $s_{K,0}$ is reached, but Adriaan van Wijngaarden discovered that not only is it faster, but oftentimes more accurate, to stop 2/3 of the way through. This means that for 42 rings, rather than using $s_{42,0}$ as the converged value, $s_{28,14}$ should be used.

3.5 Domain Size

With the boundary treatment and initial condition decided, domain sizes for the CFD comparison cases have to be selected. The size of the 'box' around the vortices is known to influence the solution, but exactly how *much* is unknown. Of the three parameters being studied in this work (with the other two being polynomial degree and cell density), it is in many ways the least predictable and yields the most noticeable changes in the solution. It's obvious that the larger the size of the box, the closer the solution should be to the unbounded result, but since computational time and power limit what can practically be modeled, the goal is twofold: determine a domain size that is an acceptable approximation of the unbounded case, and determine what the effect is to use a smaller domain if needed.

Given that the Lagrangian results are ultimately intended for comparison with CFD results, the dimensions of the various box sizes are chosen accordingly. For instance, thinking ahead to the CFD, it does not make sense to waste a lot of resources modeling a very wide domain, when Figure 3.5 proves the vertical direction is the one of interest. Thus, a tall and skinny rectangular domain is a better use of domain space than a square domain since it doesn't unnecessarily model domain that won't be used. However, it is common practice in CFD to only model half of the problem when the geometry is symmetric because it allows accurate modeling of the problem with only half the cells (and therefore half the time). This

means that the tall and skinny box for the CFD will only encompass two vortices and needs to be doubled in width to obtain the corresponding Lagrangian domain sizes.

With Figure 3.5 as a guide, two initial box sizes were selected: one intended to be highly restrictive and another to allow the vortices to convect more freely. The smaller size, which is denoted as 'small' from here forward, places the lower boundary at $z = -32$ and the right boundary at $y = 40$. It is expected that the solution will be much different from the unbounded case, but it serves as an interesting case numerically and also approximates the proximity of wind tunnel walls in experimental results. Figure 3.7 depicts the behavior of the vortices in the Small domain for two revolutions, where red outlines the Lagrangian domain size and the dashed blue line divides it to show the corresponding CFD domain. As expected, the lower boundary significantly impedes the downward motion of the vortex pairs and even pushes them to the side.

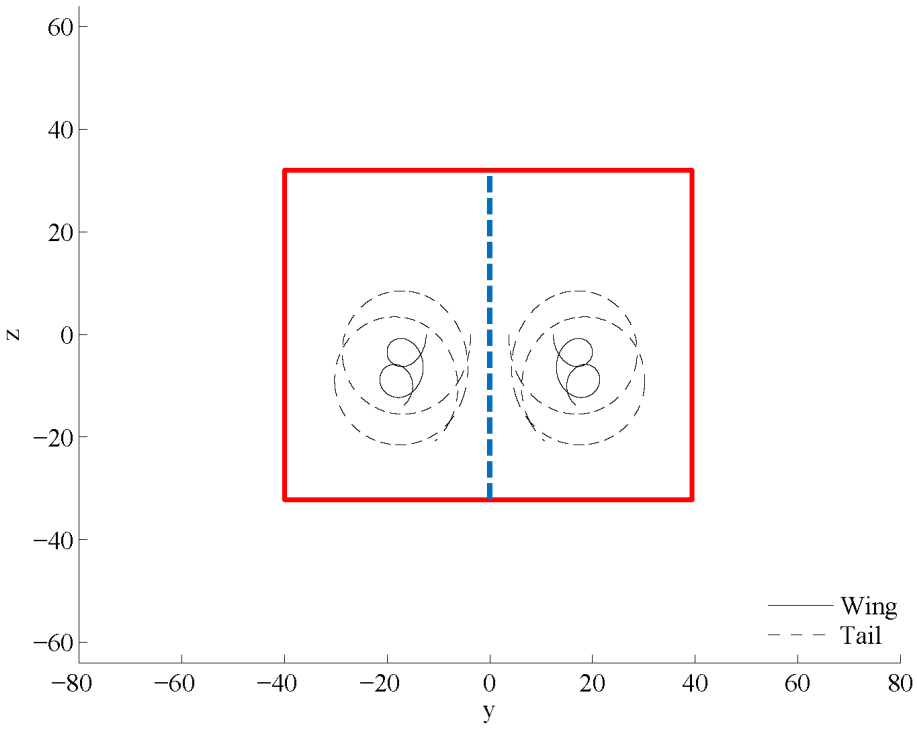


Figure 3.7: Small domain

The dimensions of the domain for the 'small' box are then doubled to obtain the second size, which will be called 'medium'. This puts the lower boundary at $z = -64$ and the right boundary at $y = 80$. Two revolutions are shown in Figure 3.8, where the medium box clearly provides much more space around the vortices. The pairs are able to convect downward and are not pushed apart. However, a close look reveals that even though the boundary appears to be well away from the vortex trajectory, it still prevents the pairs from moving down as far as in the unbounded case.

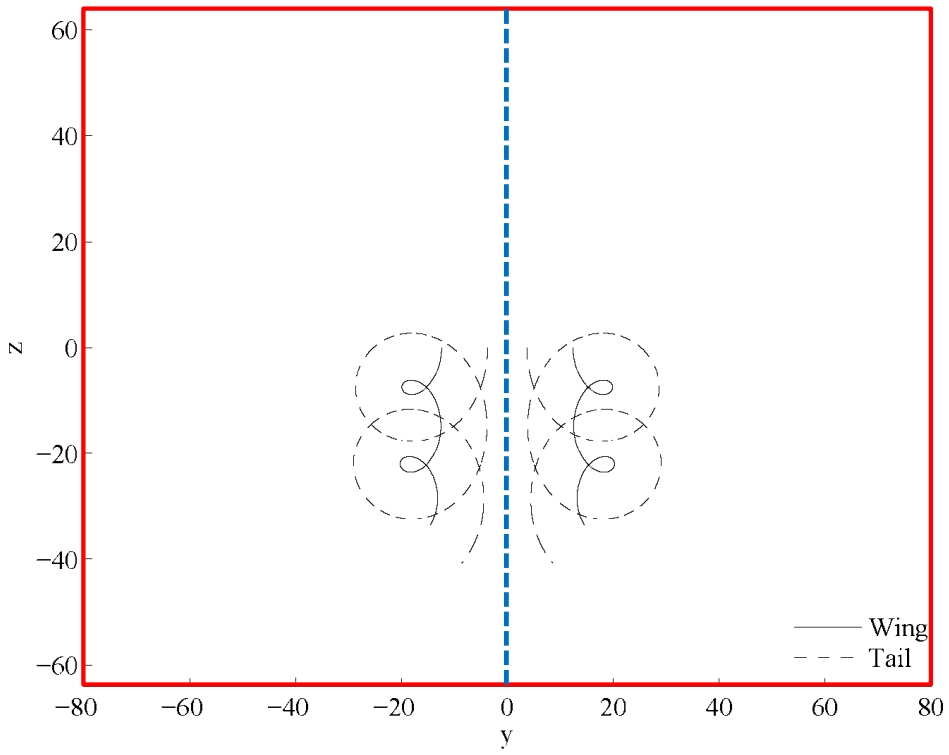
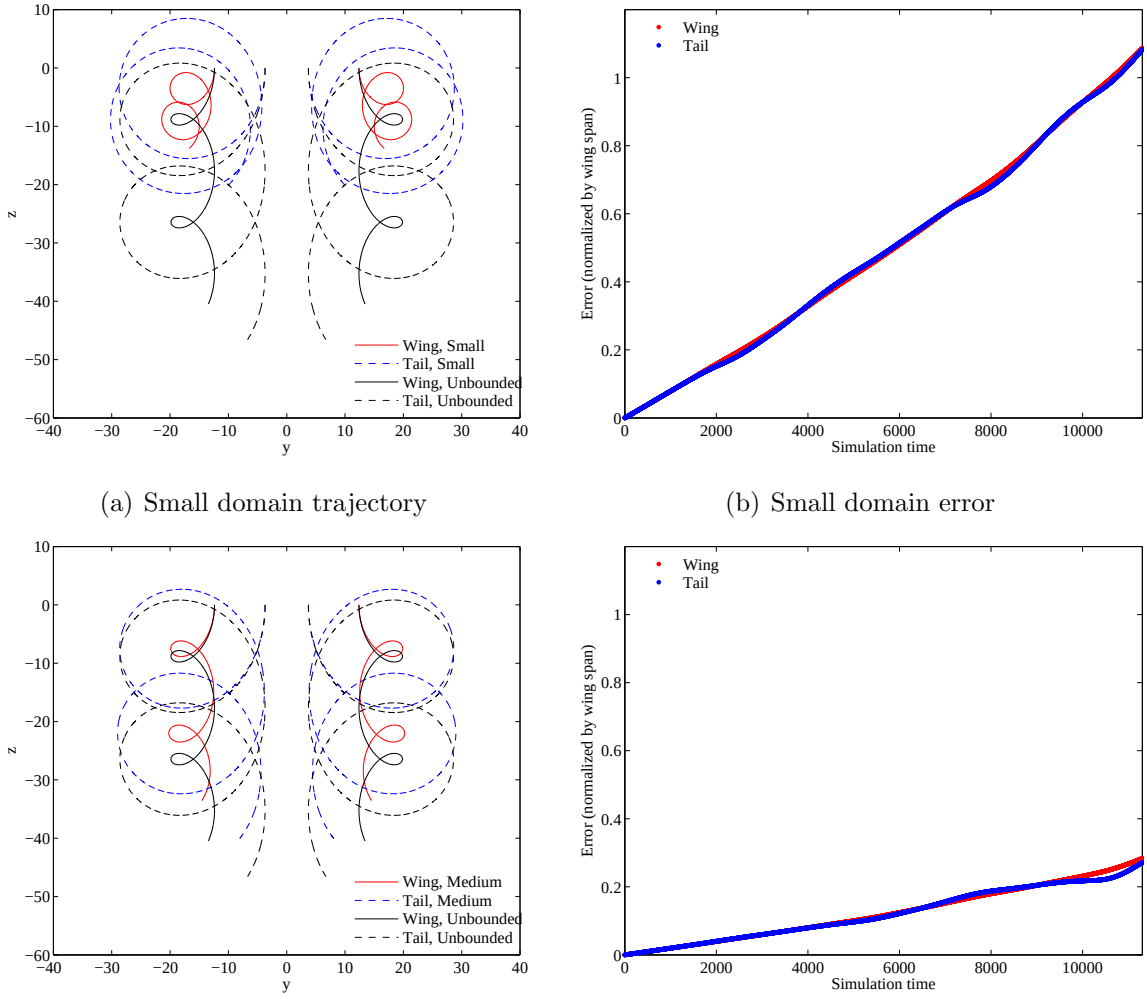


Figure 3.8: Medium domain

Figure 3.9 shows each of the bounded trajectories plotted with the unbounded case and their corresponding error, where error at any given time is calculated as the distance between the vortex centers for the bounded and unbounded trajectories. The error is then normalized by the wing span so that the distance can be thought of as a fraction of the wingspan. By the end of the two revolutions, the vortices confined to the small box are more

than a full wing span away from where they should be according to the unbounded case. The medium box has less effect on the vortex system, as the error is reduced by a factor of four and the vortices are about a third of a wing span from the unbounded results at the end of the simulation. Figure 3.10 affords a closer and less cluttered look at the details of the trajectories and plots them with the same limits as the unbounded case, regardless of where the domain boundaries are.



(a) Small domain trajectory

(b) Small domain error

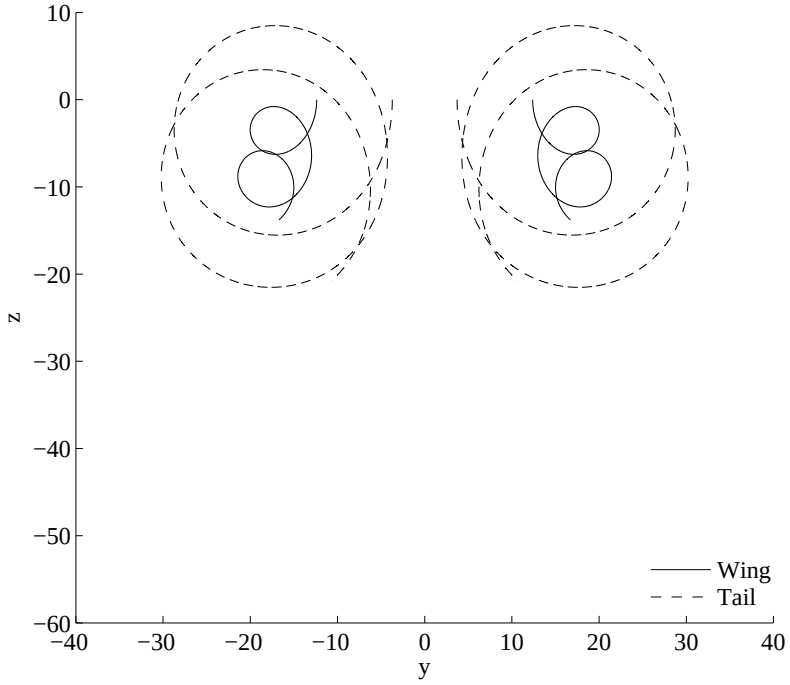
(c) Medium domain trajectory

(d) Medium domain error

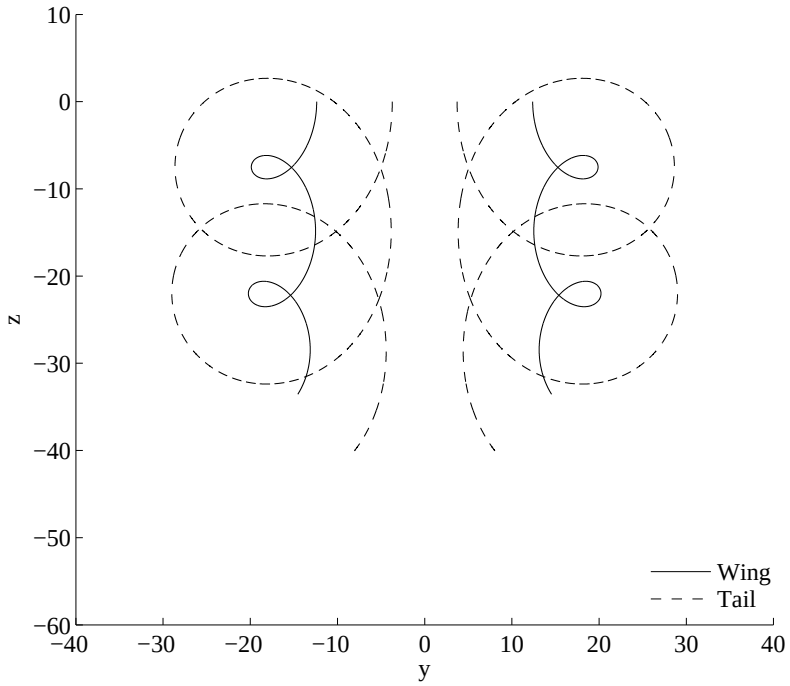
Figure 3.9: Small vs. Medium domain (compared to unbounded case)

The Lagrangian results outlined in this chapter will serve as the 'exact' results for assessing how well the DG method performs. Comparison will keep domain sizes consistent

so that any CFD results on the small domain will likewise be compared to the Lagrangian results on the small domain, and so on.



(a) Small



(b) Medium

Figure 3.10: Lagrangian solutions on bounded domains

Chapter 4

Modeling Counter-rotating Vortices with DG

4.1 Problem Setup

With the groundwork for the method laid and exact results obtained with the Lagrangian simulation, the counter-rotating vortices can finally be modeled with the DG method. Some of the same methods outlined in Chapter 3 still apply, like the vortex profile, circulation, and radius. The main difference is that instead of modeling all four airplane vortices, only one wing and one tail vortex is actively modeled, while the other two are modeled through symmetry. Modeling all four vortices with the Lagrangian approach does not add any cost, but it would be twice as expensive to do so with DG and no benefit would be gained. Therefore, using symmetry where possible is extremely important. The change is reflected in Figure 4.1, where the dashed arrows represent the vortices modeled through symmetry.

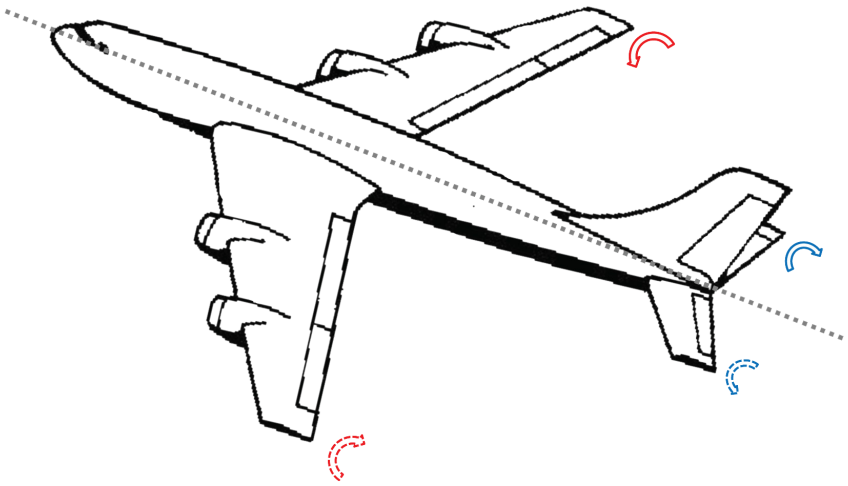


Figure 4.1: Diagram of vortices to be modeled

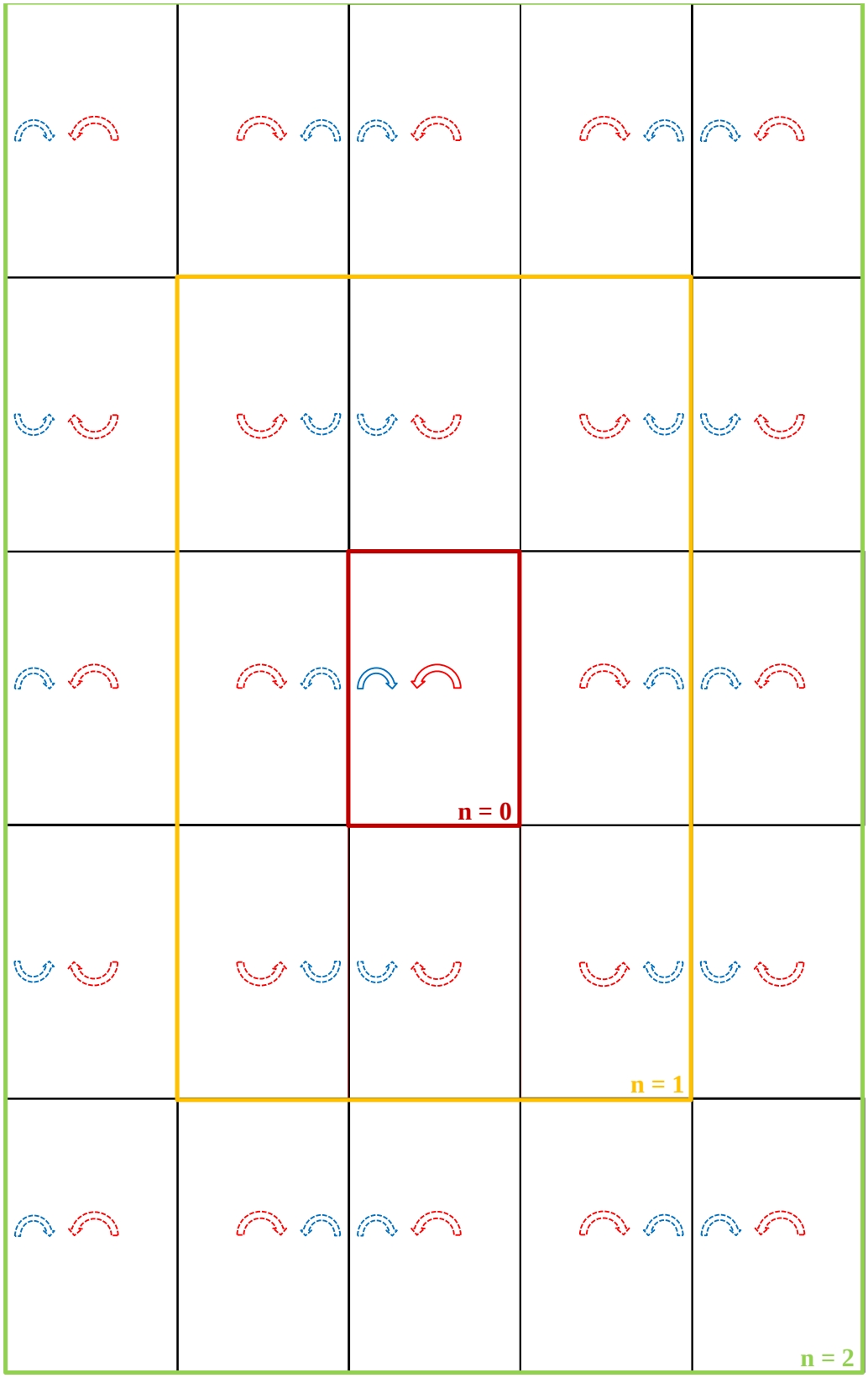
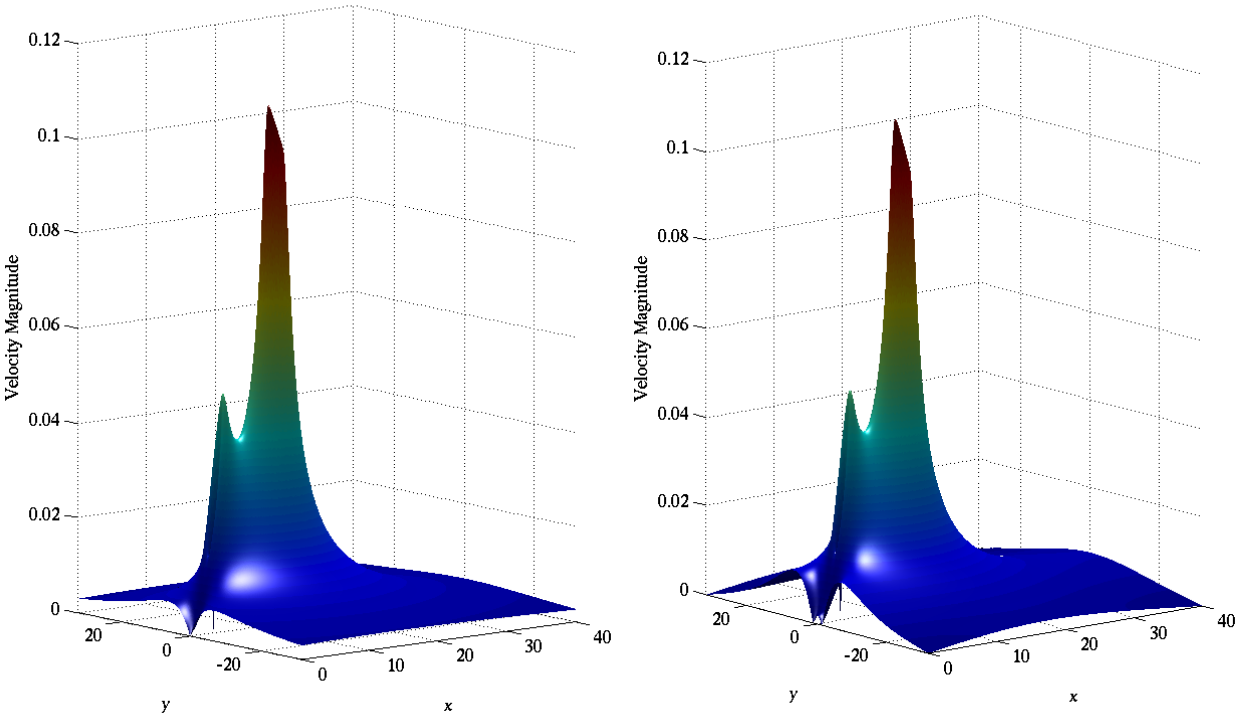


Figure 4.2: Vortex ring setup

Another part that carries over from the Lagrangian simulation is the idea of vortex 'rings' to help construct an initial condition that is consistent with the choice of reflective boundary conditions. Figure 4.2 illustrates that with the Lagrangian domain divided in two, the vortices are now off center and placed next to the left boundary. Again, the effects of 42 rings are added to the velocity fields produced by the two vortices in the actual domain (ring 0) and accelerated with Van Wijngaarden acceleration to obtain the total effect. The major result of the initialization is that it drives the velocities to 0 at the corners, since the only way to accommodate four vortices rotating in different directions is for the velocity to be 0. Figure 4.3 demonstrates this effect, where 0 rings in (a) is represents a case when reflective boundaries are not accounted for when creating the initial condition and nonzero velocities occur at the corners. By incorporating the effects of the mirrored vortices, Figure 4.3 (b) is believed to be a better and more consistent starting point for the simulation.



(a) 0 rings

(b) 42 rings

Figure 4.3: Impact of vortex array on initial conditions

Unfortunately, the only problem with adding the perturbations of the vortices in the rings is that only the *velocities* can legally be added - adding the pressures and densities yields non-physical results because they are nonlinear. Thus, the velocity field obtained through the initialization is the only part that can be used, and the other values have to be produced by another method.

To rectify this situation, the initial state is replaced with $[\rho, \rho u, \rho v, \rho e_t] = \left[1, 0, 0, \frac{1}{\gamma(\gamma - 1)}\right]$ and the velocity field obtained with the vortex rings is used as a source term and introduced slowly with a scale factor using a DG scheme. Figure 4.4 shows that the scale factor starts at 0 and is gradually increased in a cosine ramp until it reaches 1 at a simulation time of 100. After that, it is held constant, and the velocities are held to those of the source term while the pressure and density values gradually settle to their physically correct values. The result is an entire initial condition that agrees with the reflective boundary conditions without making any assumptions that the boundary is far enough away to be negligible.

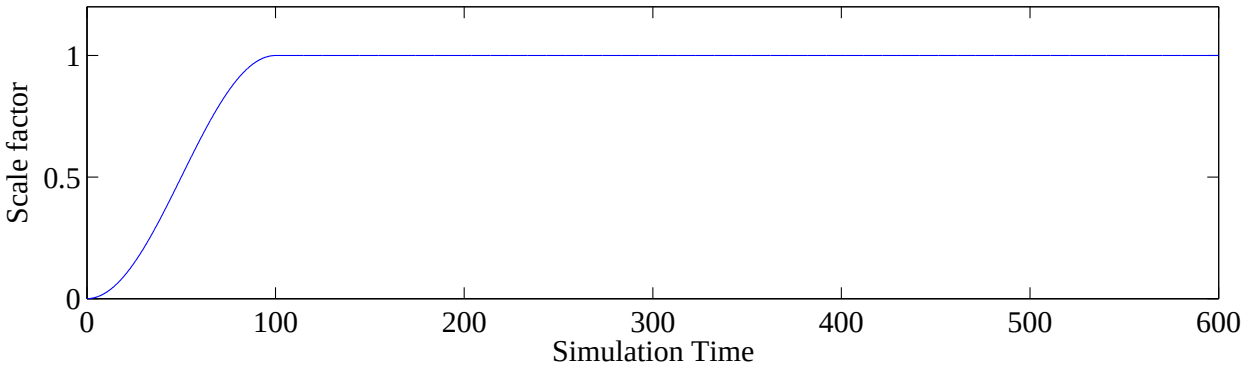


Figure 4.4: Scale factor ramp during initialization

Carrying this procedure out to a time of at least 600 ensures that the residuals drop from their maximum value at least three orders of magnitude, at which point the state is stable and ready for the full simulation. The residuals for the initialization of one case are shown in Figure 4.5, where the normalized residual peaks at 7.59×10^5, but falls to 722 by the end of initialization. Even just by the end of the source term ramp, the residuals come down more than two orders of magnitude.

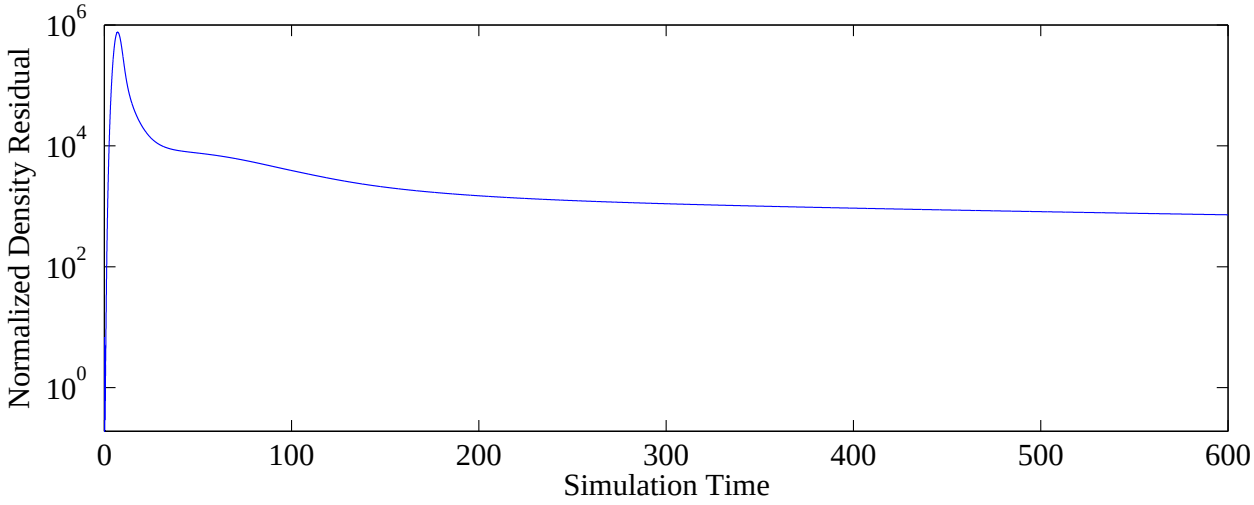


Figure 4.5: Sample residuals during initialization

4.2 Selecting Parameters

The next step is to run the actual DG simulation of the vortex interaction, but before that can happen, some preliminary case-defining parameters have to be set. As mentioned previously, these are primarily domain size, polynomial degree, and cell density (the latter two of which constitute grid resolution). A few options for each are chosen so the various combinations of parameters can give insight into how they affect the accuracy and performance of the DG method.

During the setup of the Lagrangian cases, box size was considered and defined for two separate sizes, which were termed 'small' and 'medium'. The DG simulations are also conducted on these domains so they can be compared directly. They are rectangular to allow as much downward movement as possible, with the small box's dimensions being [0,40] x [-32,32] and the medium's being [0,80] x [-64,64].

With box sizes defined, the next parameter to be decided is polynomial degree. Low, moderate, and high polynomial degree are covered by using degrees of 3, 9, and 14. In tandem with polynomial degree, cell density is used to determine how many cells should be

used to represent the domain. Cell density is controlled through degrees of freedom (DOF) per unit, where DOF/unit $= d + 1/\Delta x$ and d is polynomial degree and Δx is the cell size. By controlling cell density instead of cell count, the results can more readily be compared across varying degrees and domain sizes. Much like with box size, two densities are chosen for comparison: 7.5 and 15 degrees of freedom per unit in the x-direction.

The process used to create the four basic grids is outlined in Figure 4.6. The two grids on the left represent the 'coarse' cell density (7.5 DOF/unit) and the two on the right represent the 'fine' density (15 DOF/unit). Similarly, the top two are the 'small' box size and the bottom two are the 'medium' size. By taking a look at the time step restriction from Equation 2.35, which is restated here:

$$\Delta t = \frac{C\Delta x}{\lambda_{\max} d^2}$$

one can see that doubling the cell density (the same as halving Δx) requires halving the time step, but also quadruples the number of cells for any given domain. The result is an approximate 8x increase in computational time to refine the cell density. Similarly, doubling the domain also quadruples the number of cells and computational time, but since the cell size does not change, neither does the time step and it is only about 4 times more costly. It reiterates why the ability to use larger cells is so important - huge time savings exist when larger cells can be used by applying a high-order polynomial.

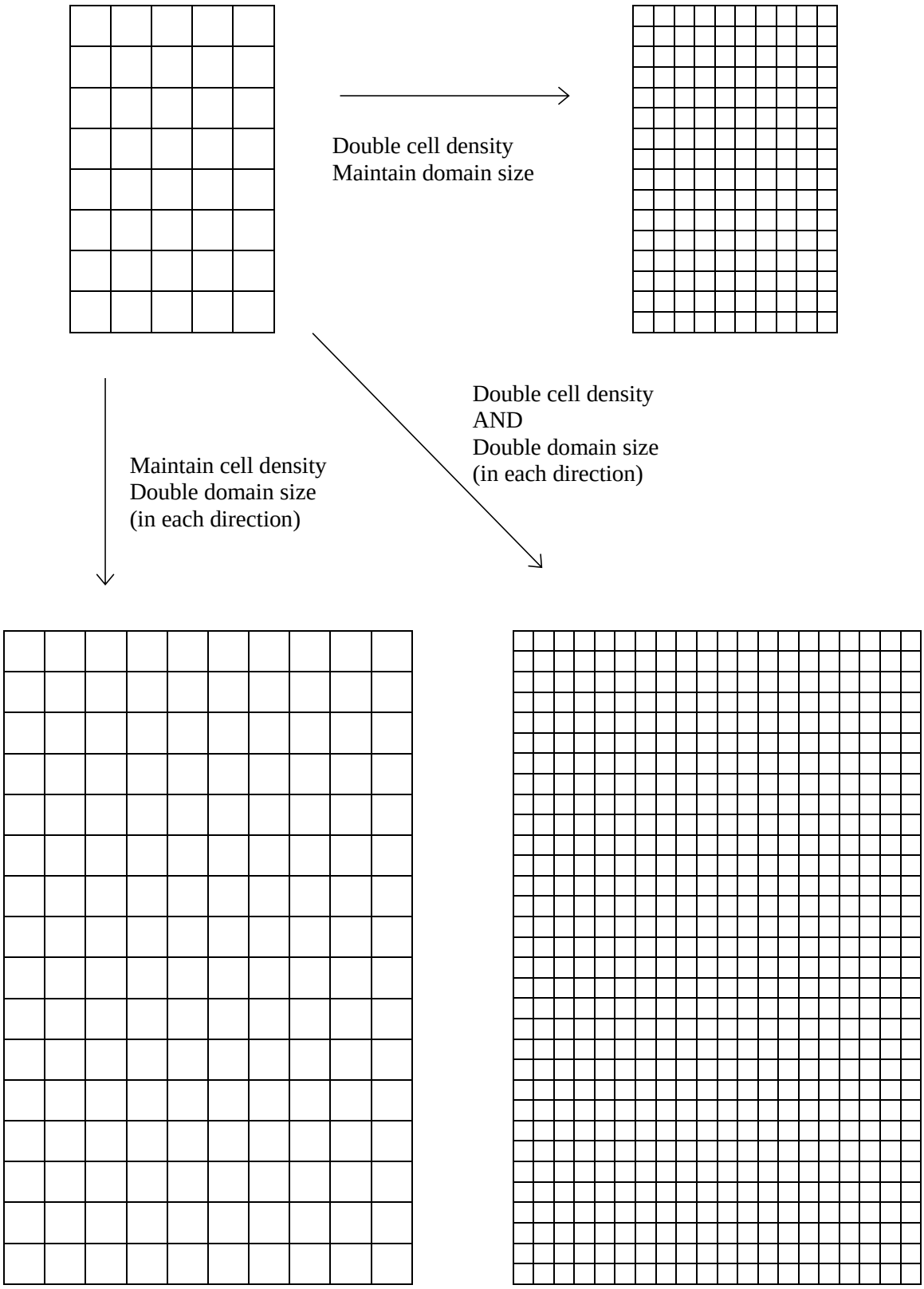


Figure 4.6: Domain sizing. Cell size and cell count are representative only.

With the above parameter selections, 12 preliminary cases are defined and outlined in Table 4.1, where small and medium refer to the domain size and the two columns under each are the coarse and fine cell densities. Since degrees of freedom are held constant for each column, fewer cells are required for the higher degrees. In fact, this is one of the biggest benefits of DG over the FV method. Looking at the 9^{th} degree case for the 'small coarse' grid, only 1440 cells are needed to obtain 144,000 degrees of freedom (in 2D), while FV would have required 144,000 cells - 100 times more than DG. A smaller-scale example is depicted in Figure 4.7, which shows a generalization of cell counts for a fourth-degree DG method compared to a FV method. The data points provided by the high cell count in FV are recovered by the polynomial nodes in the DG method.

Table 4.1: Grid sizes for test cases

Degree	Small		Medium	
3	75 x 120	150 x 240	150 x 240	300 x 480
9	30 x 48	60 x 96	60 x 96	120 x 192
14	20 x 32	40 x 64	40 x 64	80 x 128

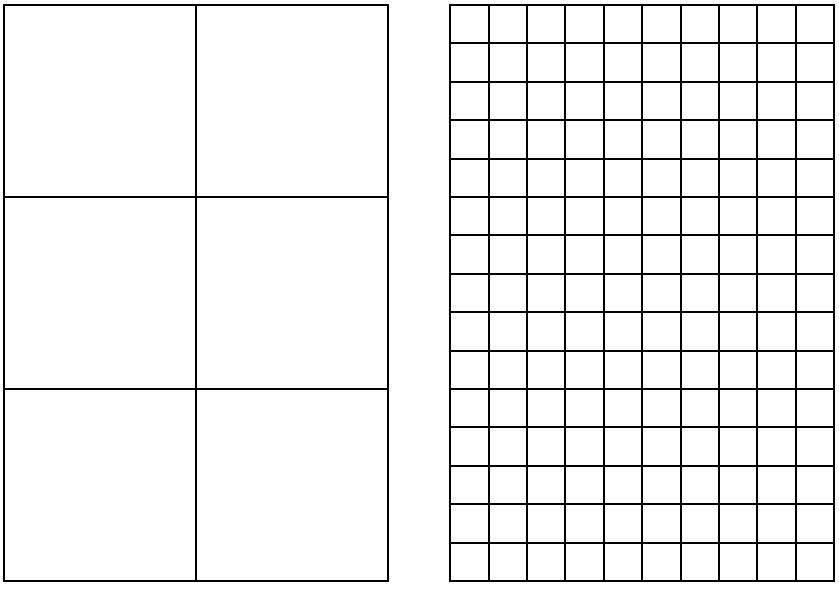


Figure 4.7: Cell count comparison for DG vs. FV to obtain same DOF

Chapter 5

Results

Since the primary focus of this research is to determine the effect of certain parameters on the solution, the results are organized and separated by parameter that was studied, so that the effects are more readily apparent with each parameter evaluated individually. The first parameter presented is polynomial degree, then domain size, followed by cell density. The effects are evaluated by comparing plots of trajectory error and kinetic energy and assessing any differences. Each grouping of plots is organized so that whatever parameter is being investigated is changed while moving down the page and the data is plotted with the same limits wherever possible to make comparison easier. Since there are three parameters, each group of plots shows the effect of changing one parameter while the other two remain the same. The captions for each group make it clear by stating the changing parameter first and putting the two that stay the same in parentheses. For instance, a caption of 'Degree 3 vs. 9 vs. 14 (Small domain, Coarse grid)' means that degree is varied, but domain size and cell density are the same for the three plots.

Each plot is labeled with a unique identifier where d- - is the degree and h- - is the cell count in the y direction (so d03 h75 is the degree 3, 75x120 case). The first set of plots presented is always trajectory error, and denotes the distance between the DG vortex center and the Lagrangian vortex center at any given time for the same case (so a DG case on the small domain is always compared to a Lagrangian case on the small domain). As in Chapter 3, the error is then normalize by wing span.

Tracking the vortex trajectory for the CFD solutions requires locating the vortex centers, which are the points where the highest and lowest (most negative) vorticity occurs. The highest vorticity corresponds to the wing vortex and the lowest corresponds to the tail

vortex, where vorticity ω is defined as

$$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{5.1}$$

In order to get $\frac{\partial v}{\partial x}$ and $\frac{\partial u}{\partial y}$, expressions for the derivatives of ρu and ρv have to be rearranged as follows:

$$\frac{\partial}{\partial x}(\rho v) = \rho \frac{\partial v}{\partial x} + v \frac{\partial \rho}{\partial x} \tag{5.2a}$$

$$\frac{\partial v}{\partial x} = \frac{1}{\rho}\left[\frac{\partial}{\partial x}(\rho v) - v \frac{\partial \rho}{\partial x}\right] \tag{5.2b}$$

$$\frac{\partial}{\partial y}(\rho u) = \rho \frac{\partial u}{\partial y} + u \frac{\partial \rho}{\partial y} \tag{5.3a}$$

$$\frac{\partial u}{\partial y} = \frac{1}{\rho}\left[\frac{\partial}{\partial y}(\rho u) - u \frac{\partial \rho}{\partial y}\right] \tag{5.3b}$$

Then, the two terms on the right of Equations 5.2b and 5.3b can both be recognized as part of the spatial derivative of the state $q = [\rho,\, \rho u,\, \rho v,\, \rho e_t]$. In order to get the derivative of the state with respect to x and y, all that remains is to apply the DG method for a spatial derivative, which results in:

$$\frac{\partial q}{\partial x} = \mathcal{J}^{-1}\mathcal{M}\left[-S^T u + L_b u^* - L_a u^*\right] \tag{5.4}$$

The general formulation of Equation 5.4 is very similar to the method outlined in Chapter 2, except instead of a numerical flux, a simple average, u^*, of the two adjacent states is used. Also, since the flux function is not present to take the normal vector as an argument, the sign on L_a has been reversed to account for the opposite-facing normal vector.

The second set of plots for every comparison look at kinetic energy, which is used as a diagnostic because it should be conserved and stay as close to the initial value as possible if the method performs well. To make the behavior clear, the kinetic energy plots are therefore

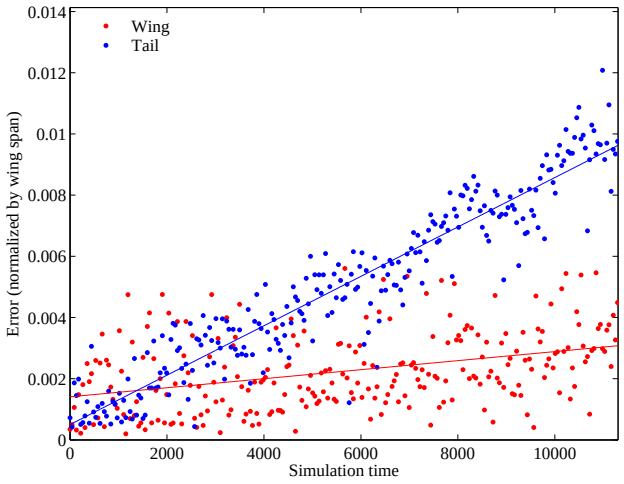
normalized by the initial value so that the values become a fraction of the original kinetic energy with a goal of maintaining 1. Kinetic energy results are also displayed as error from the initial value by considering the absolute value of the difference between kinetic energy at a certain time and the initial value and then dividing by the initial value. Once plotted on a logarithmic scale, it gives a good idea of order of magnitude differences between cases.

5.1 Degree

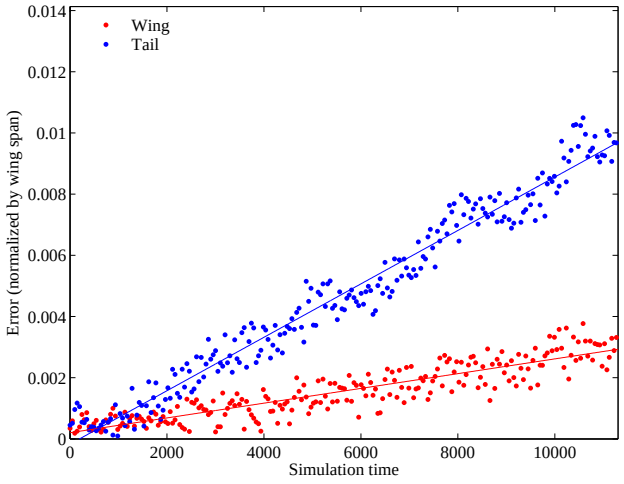
By looking at Figures 5.1 through 5.8, it can be seen that increasing the degree from 3 to 9 to 14 decreases overall error for both the wing and tail vortices, although the effect is most noticeable in the tail vortex. In the coarse grids, increasing the degree also reduces the noise in the data, especially between the degree 3 and degree 9 cases. However, the data appears to become slightly noisier and less focused again when increasing further to degree 14. This same pattern appears in the fine grids as well, but it is not as pronounced because the data is much more focused to begin with. This suggests that degree 9 may be an ideal balance between high order and computation time.

The plots of kinetic energy reveal similar trends, as degree 3 does not conserve kinetic energy as well as the corresponding degree 9 and 14 cases. The effect is most pronounced in the coarse grid cases (Figures 5.2 and 5.6), where the normalized value steadily declines and drops by about .1% (still a small amount) while the other two remain closer to 1 and hold steady. The kinetic energy error plots quantify the increase in error as approximately one order of magnitude.

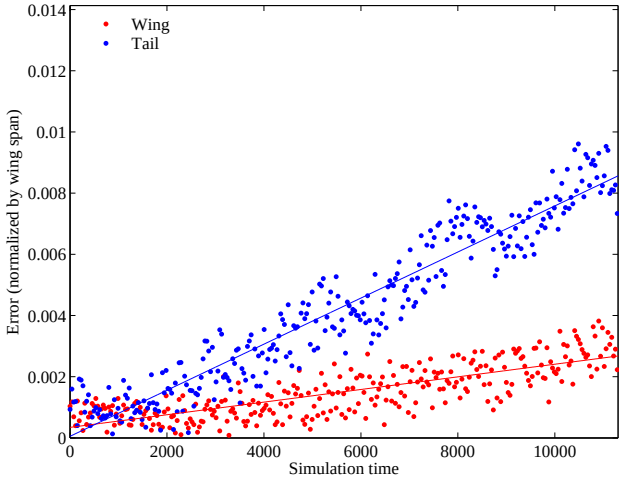
Another interesting finding is that the kinetic energy error plots seem to confirm what the error plots suggested - that at least for the coarse grids, degree 9 may be a better option than degree 14, since the overall KE error is slightly lower. The fine grids are less conclusive for determining degree's effect on the method, since the increased number of points seems to make all three degrees perform roughly the same. Degree 3 still exhibits *slightly* higher kinetic energy error, but the degree 9 and 14 errors are almost indistinguishable.



(a) d03 h75

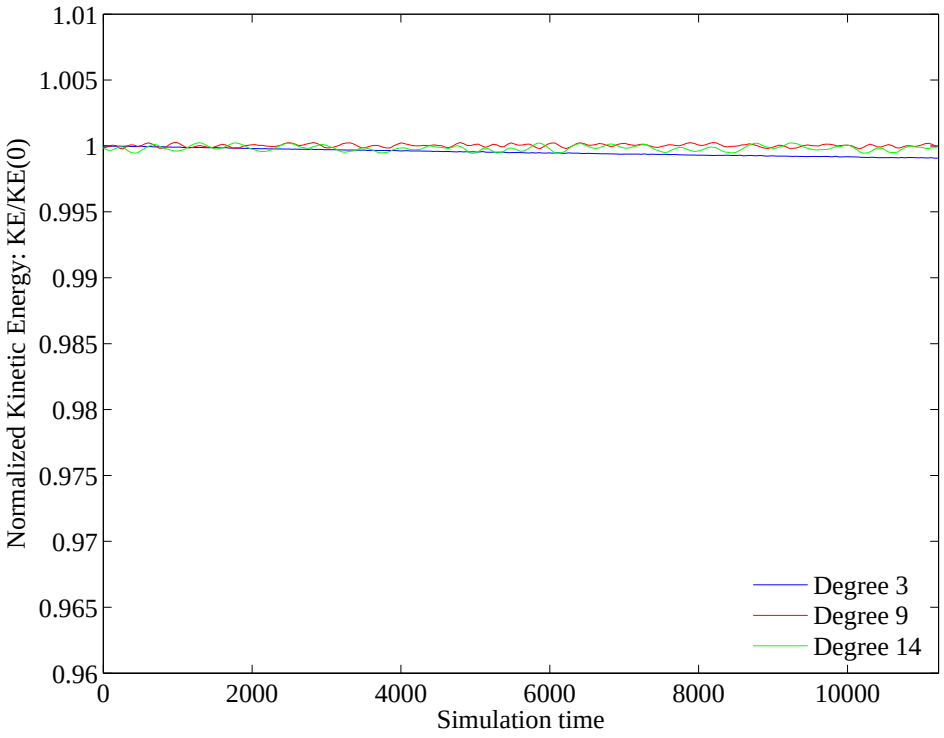


(b) d09 h30

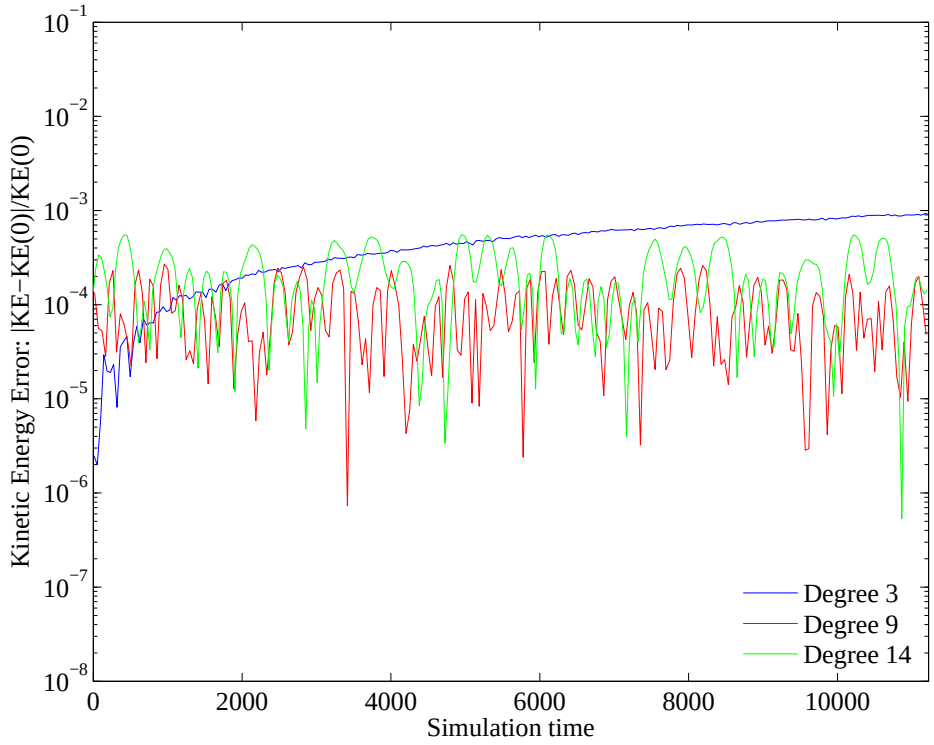


(c) d14 h20

Figure 5.1: Trajectory error: Degree 3 vs. 9 vs. 14 (Small domain, Coarse grid)

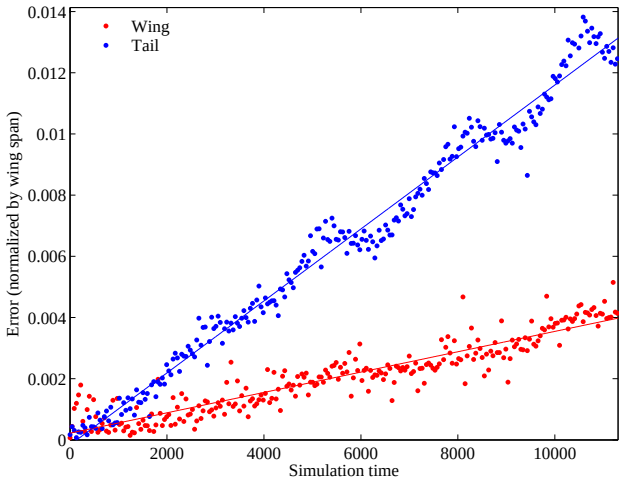


(a) Normalized Kinetic Energy

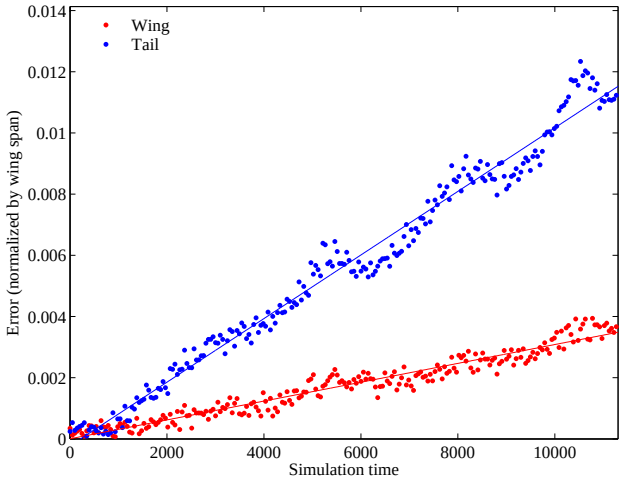


(b) Kinetic Energy Error

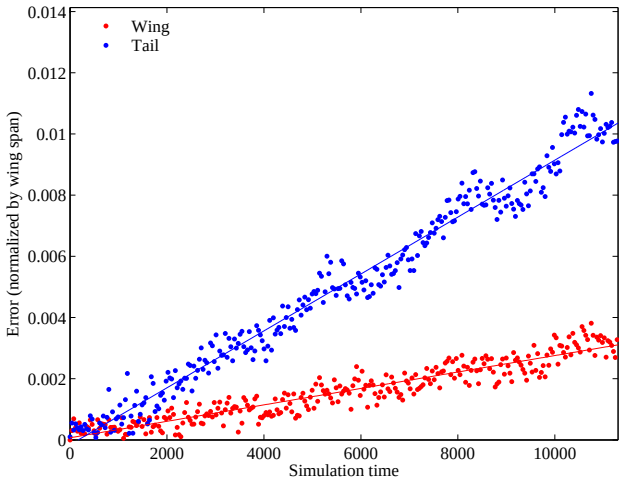
Figure 5.2: KE error: Degree 3 vs. 9 vs. 14 (Small domain, Coarse grid)



(a) d03 h150

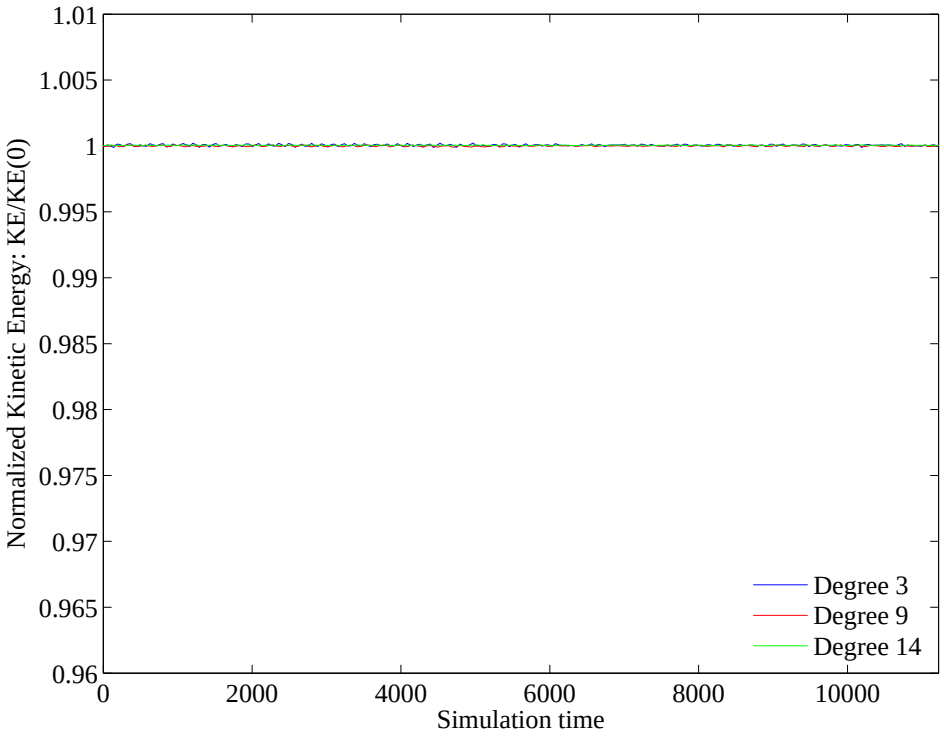


(b) d09 h60

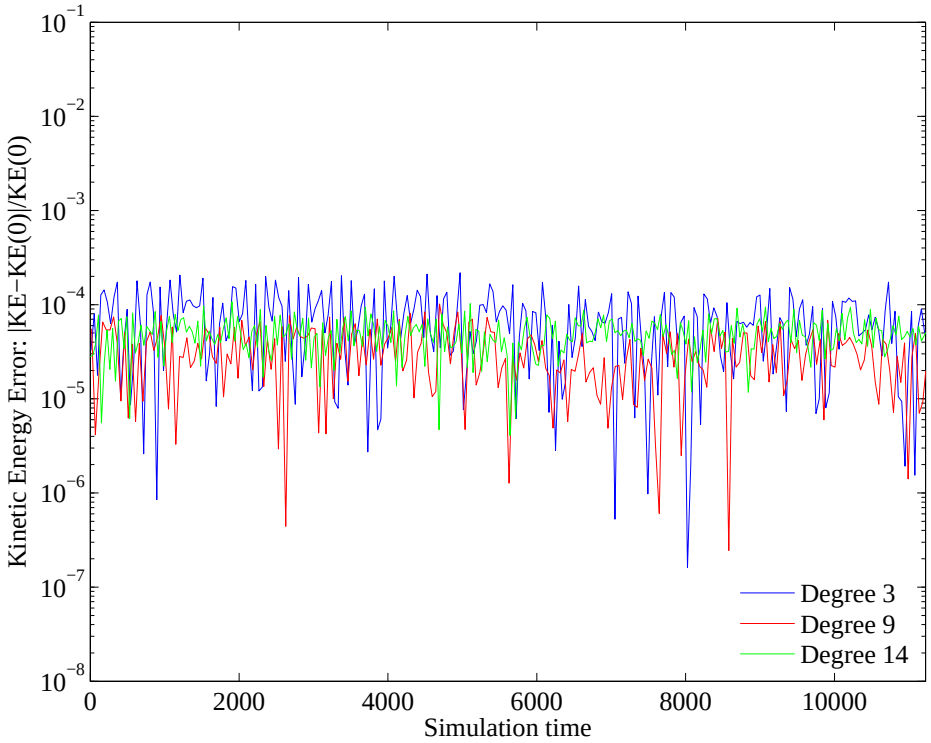


(c) d14 h40

Figure 5.3: Trajectory error: Degree 3 vs. 9 vs. 14 (Small domain, Fine grid)

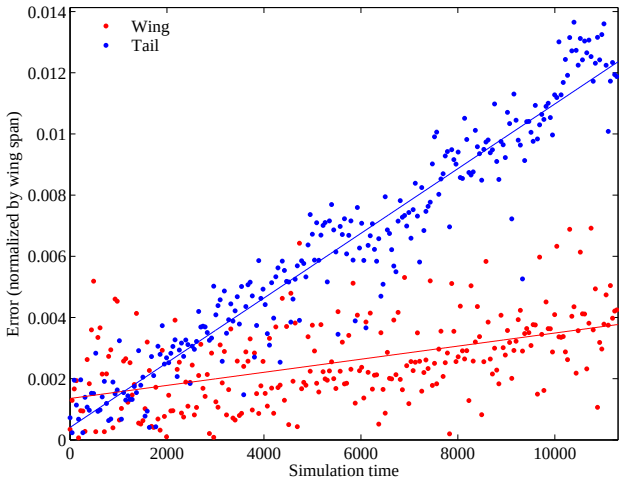


(a) Normalized Kinetic Energy

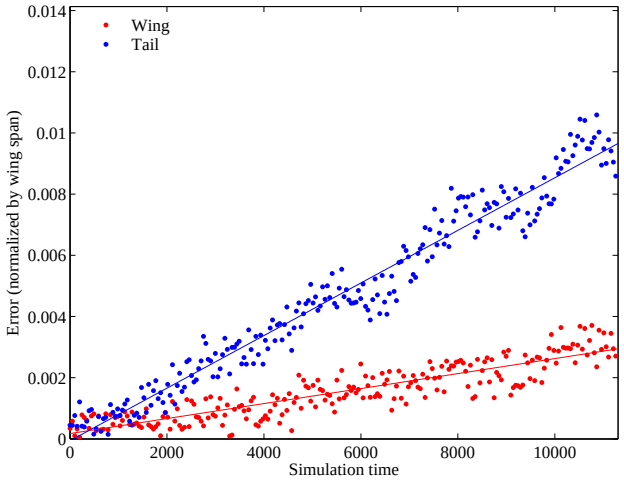


(b) Kinetic Energy Error

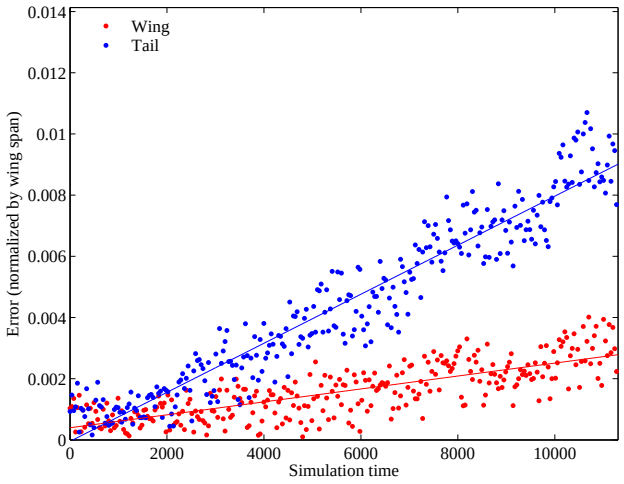
Figure 5.4: KE error: Degree 3 vs. 9 vs. 14 (Small domain, Fine grid)



(a) d03 h150

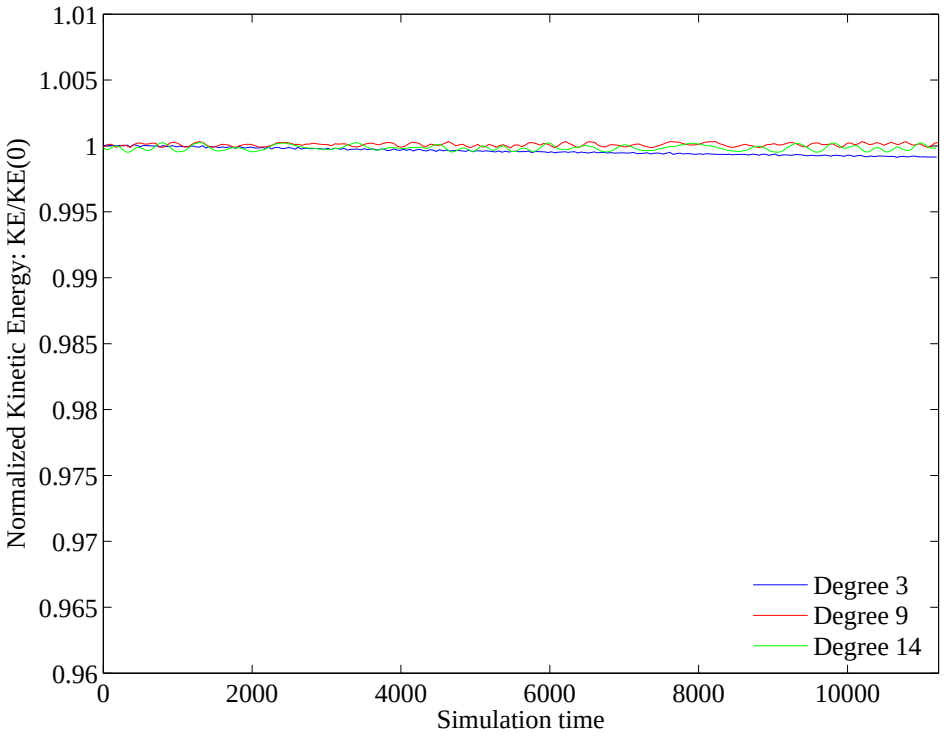


(b) d09 h60

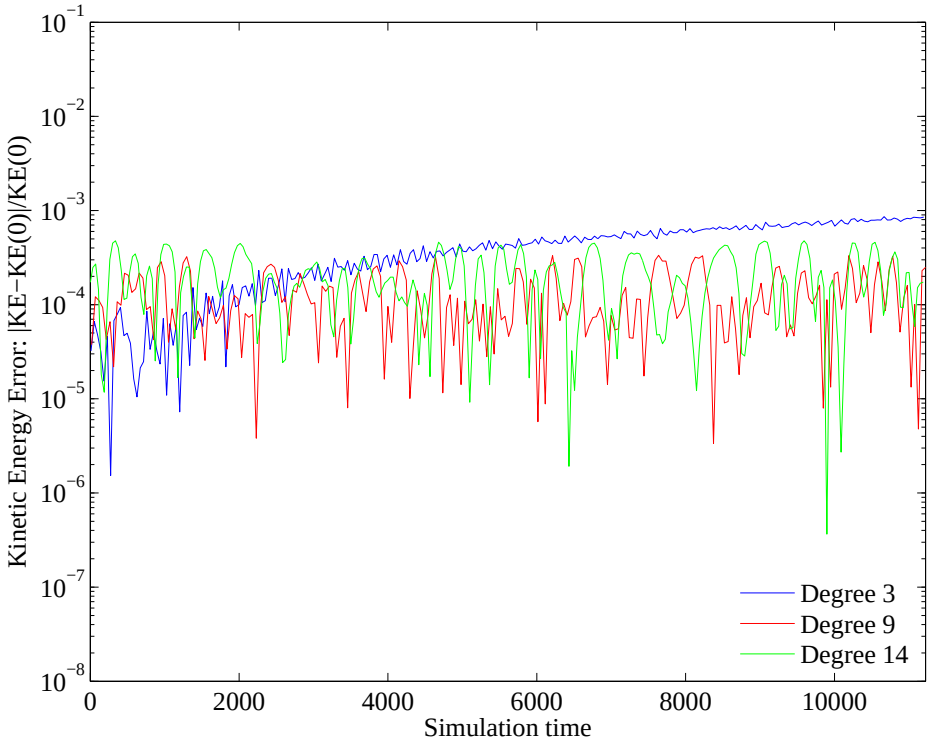


(c) d14 h40

Figure 5.5: Trajectory error: Degree 3 vs. 9 vs. 14 (Medium domain, Coarse grid)

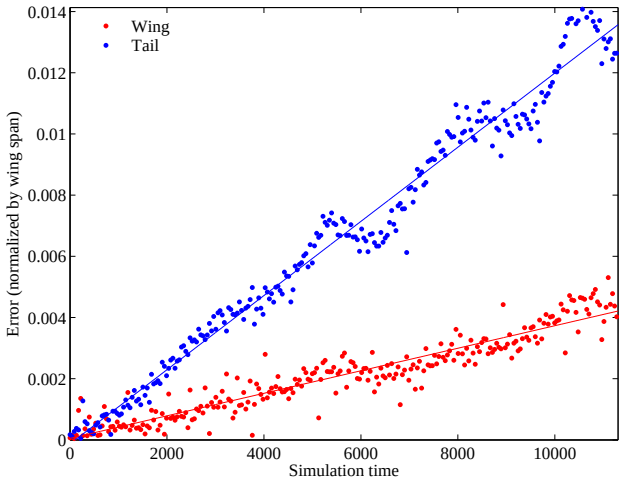


(a) Normalized Kinetic Energy

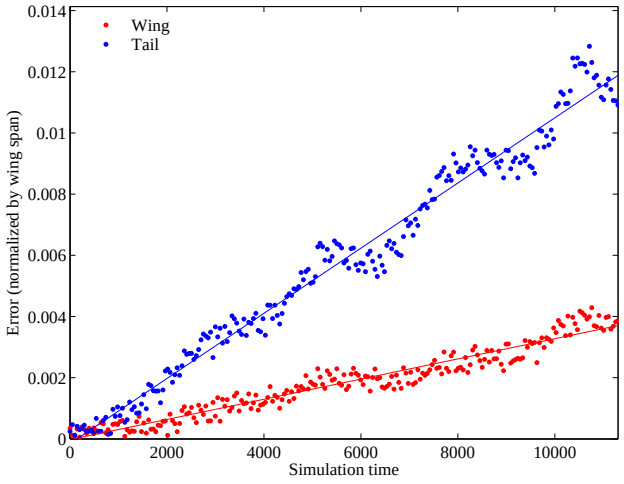


(b) Kinetic Energy Error

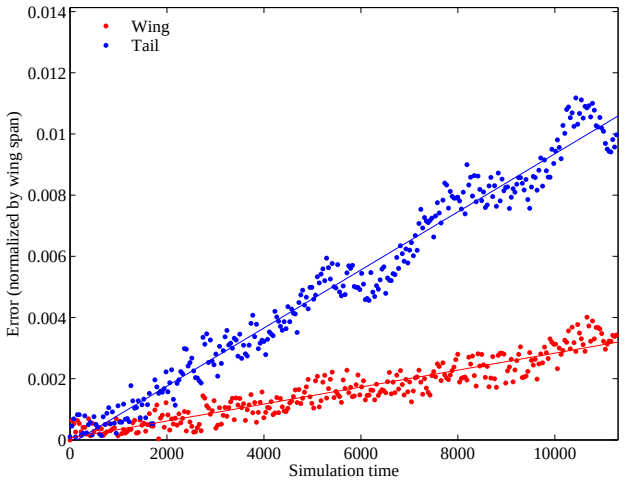
Figure 5.6: KE error: Degree 3 vs. 9 vs. 14 (Medium domain, Coarse grid)



(a) d03 h300

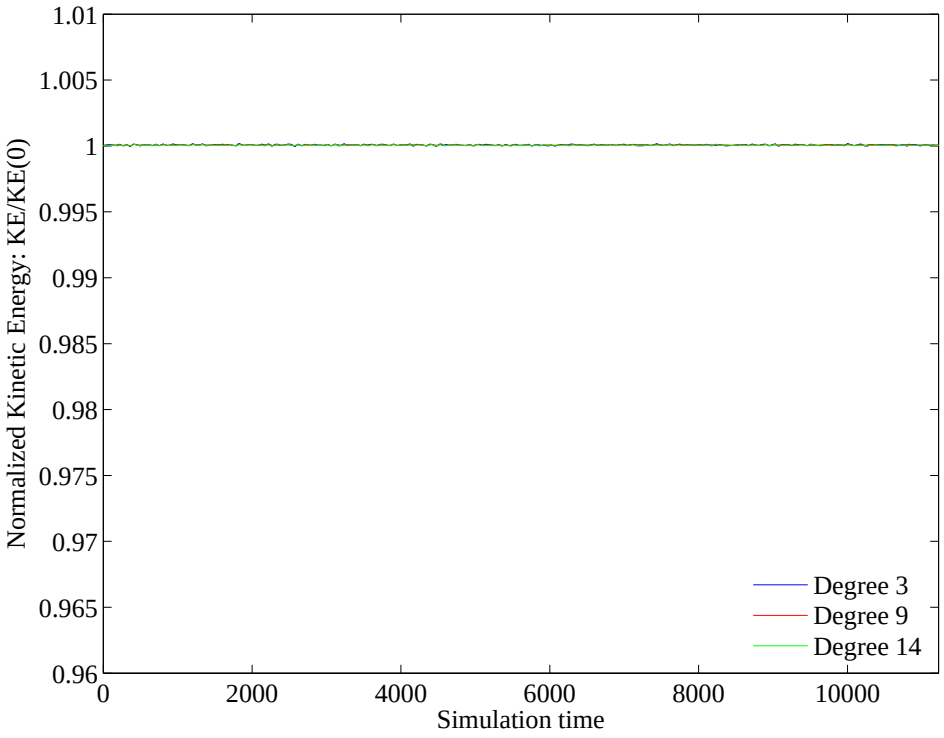


(b) d09 h120

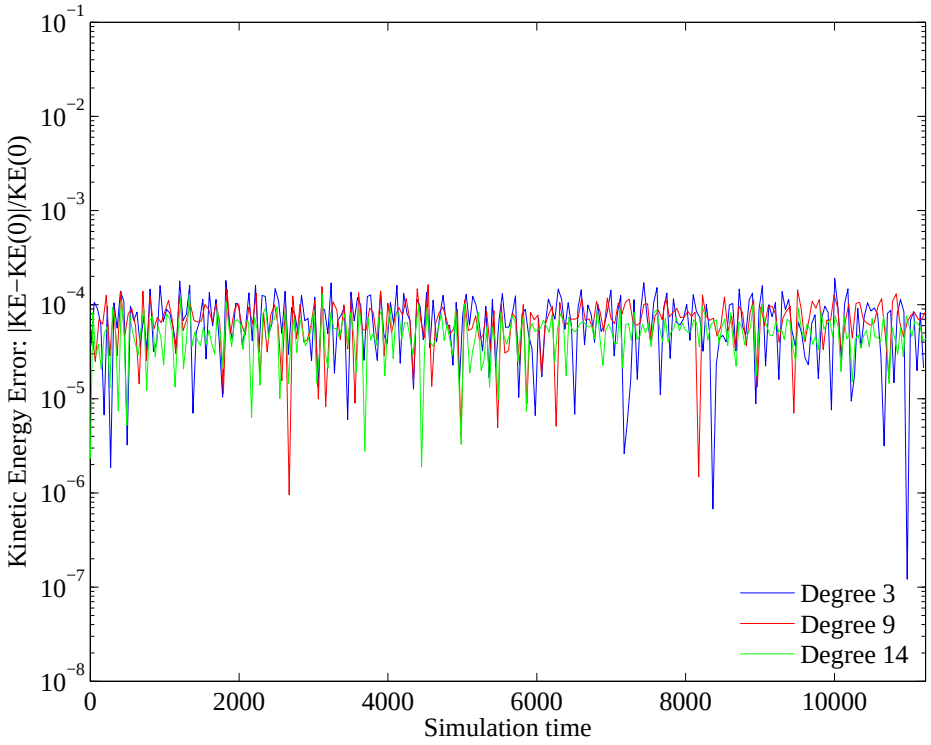


(c) d14 h80

Figure 5.7: Trajectory error: Degree 3 vs. 9 vs. 14 (Medium domain, Fine grid)



(a) Normalized Kinetic Energy



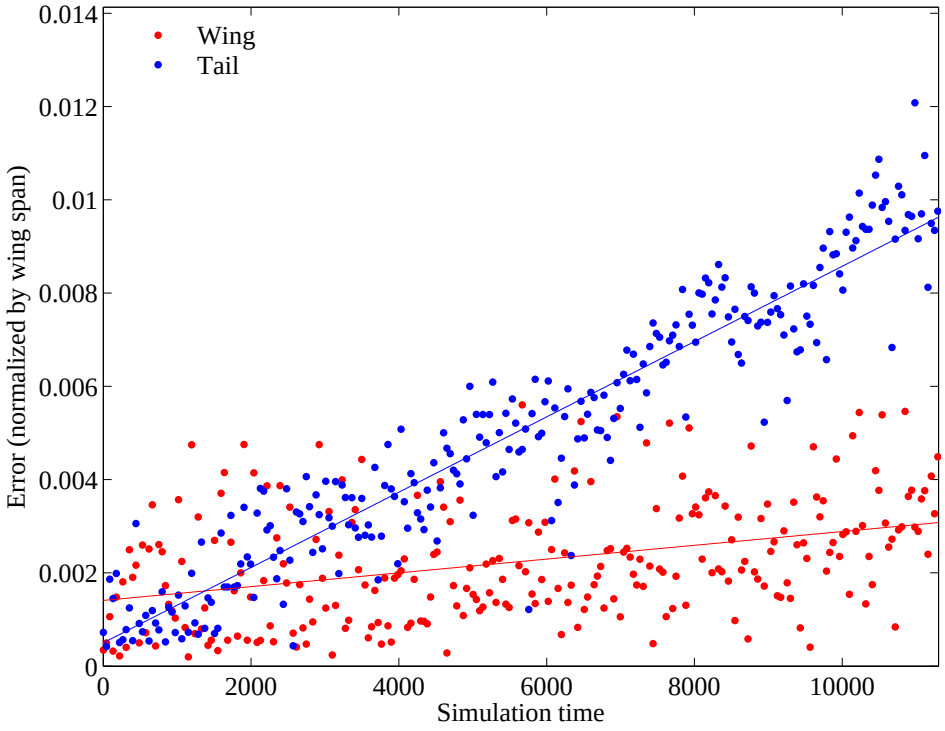
(b) Kinetic Energy Error

Figure 5.8: KE error: Degree 3 vs. 9 vs. 14 (Medium domain, Fine grid)

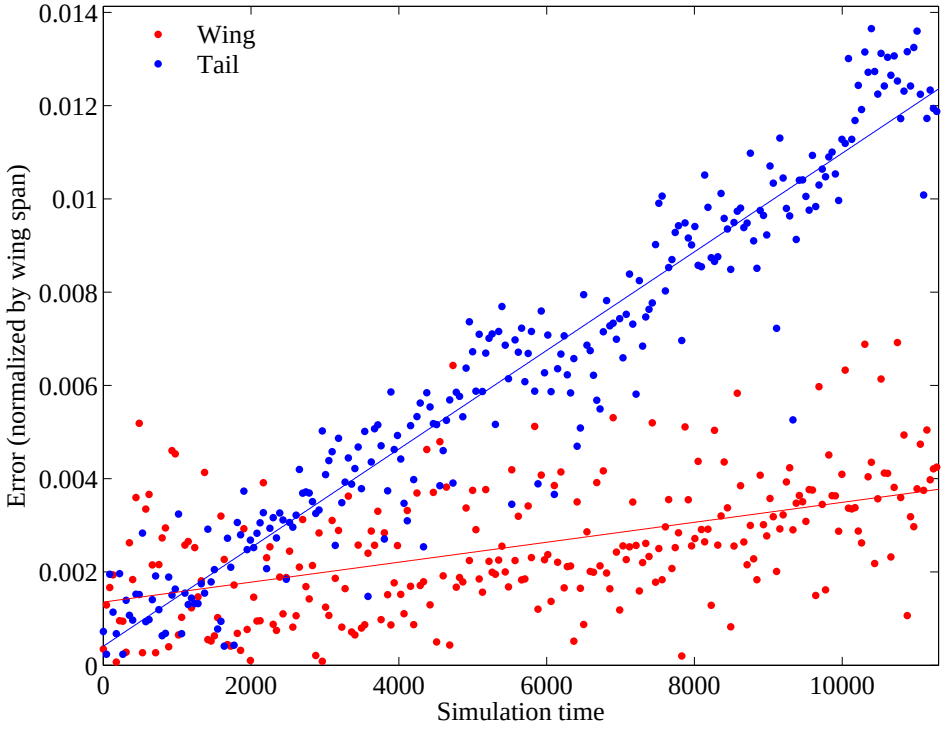
5.2 Domain Size

The effect of domain size was assessed by considering two sizes: a small domain and a medium domain (which is twice as large in both directions). The results are presented in Figures 5.9 through 5.20, which reveal that doubling the domain size has little to no effect on the error when compared to the correlating exact solution, as most of the cases exhibit almost identical error for the small and medium domains. The only exception is the third-degree coarse case, where less error is observed in the smaller domain case.

Each of the kinetic energy plots also show little or no change between cases where the only difference is domain size. If the normalized value decreases for the small case, it also decreases for the medium case, and so on. Even minute differences are almost nonexistent, as the KE error plots are remarkably similar. Taken with the trajectory error findings, the conclusion is that while boundary placement certainly affects the solution (in terms of the trajectory), it does not have any effect on the DG method itself, which performs equally well for either domain size.

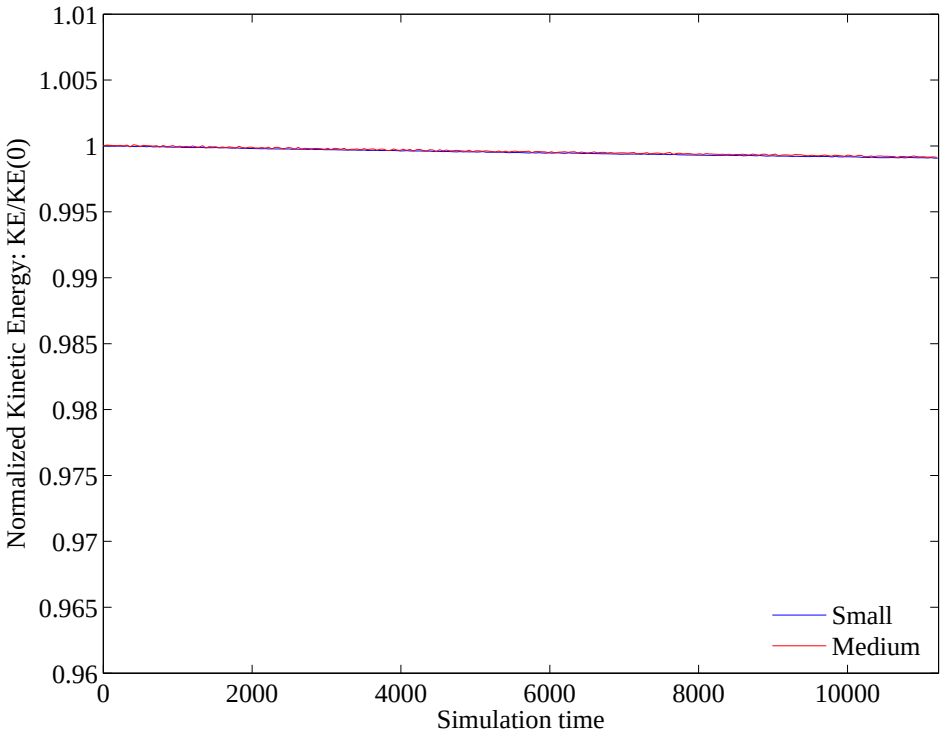


(a) d03 h75

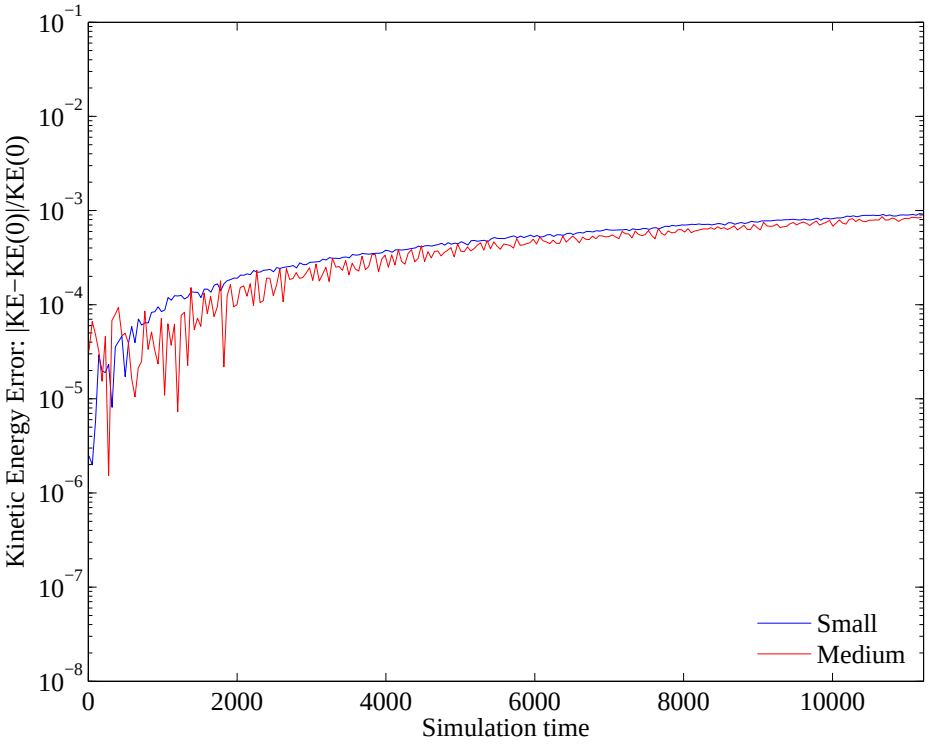


(b) d03 h150

Figure 5.9: Trajectory error: Small vs. Medium domain (Degree 3, Coarse grid)

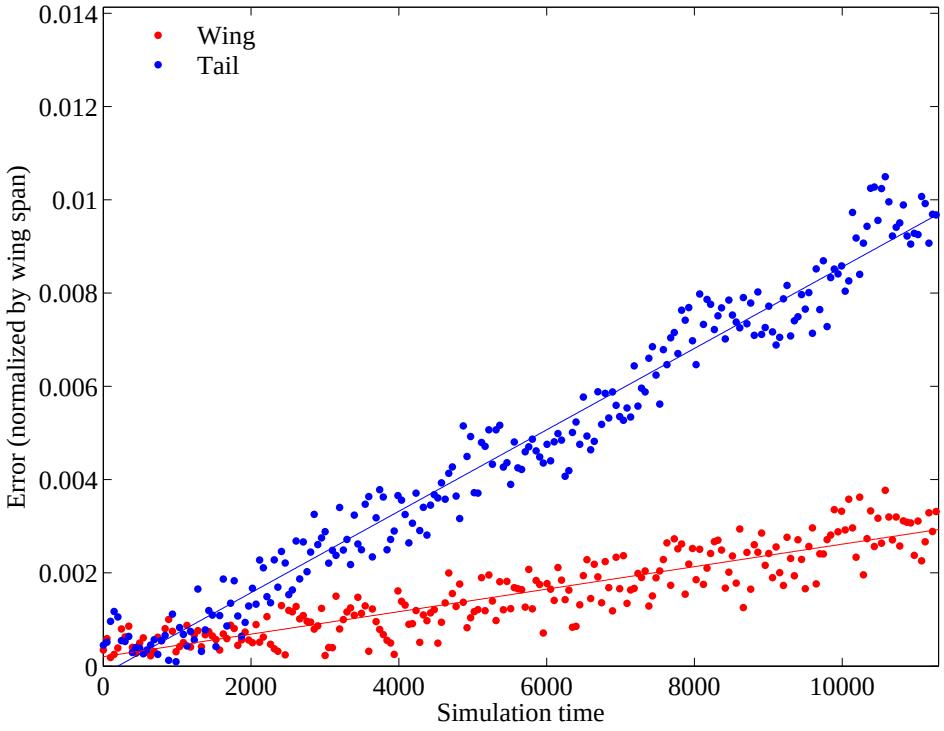


(a) Normalized Kinetic Energy

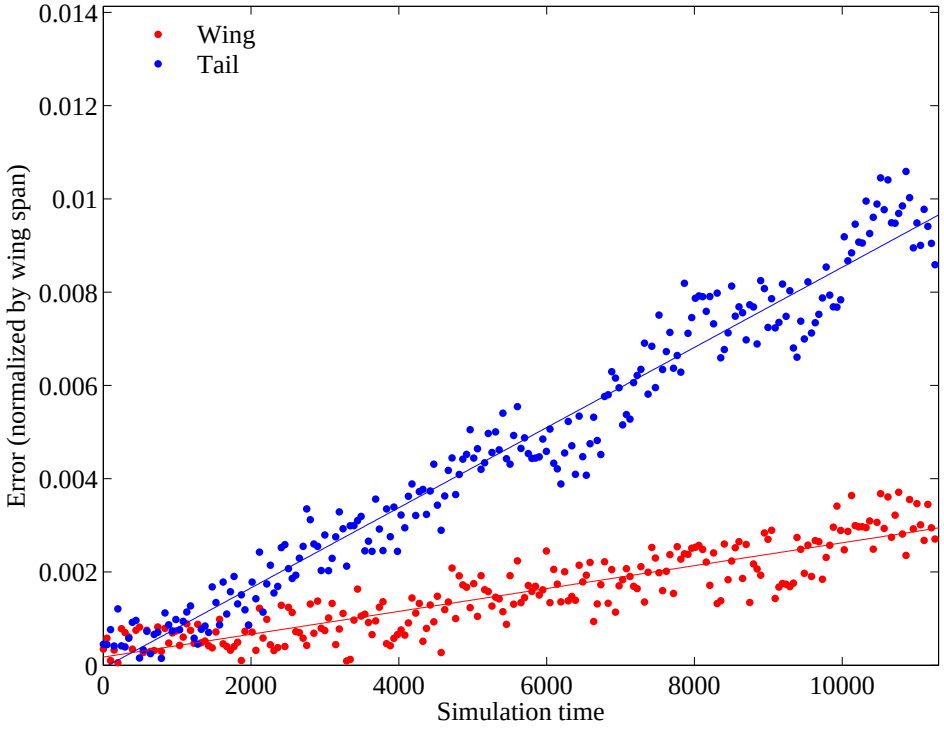


(b) Kinetic Energy Error

Figure 5.10: KE error: Small vs. Medium domain (Degree 3, Coarse grid)

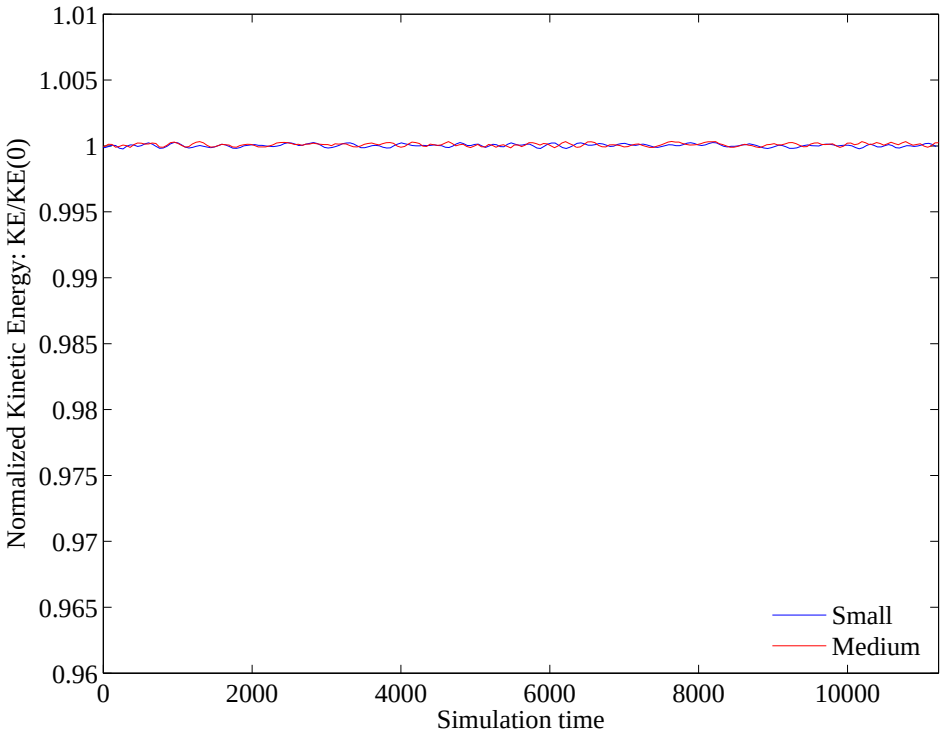


(a) d09 h30

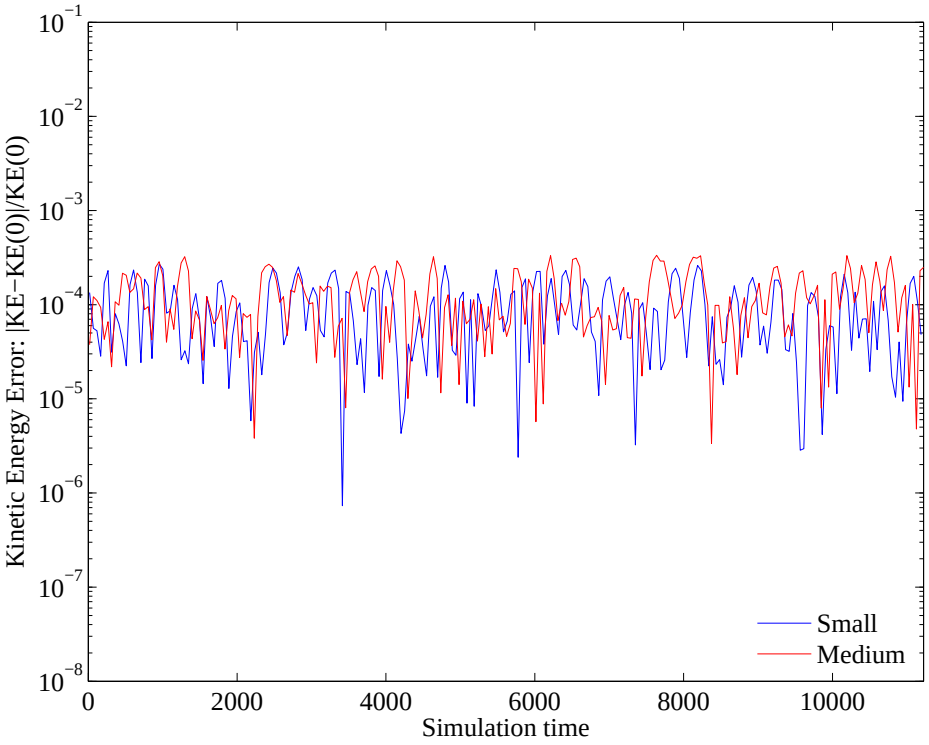


(b) d09 h60

Figure 5.11: Trajectory error: Small vs. Medium domain (Degree 9, Coarse grid)

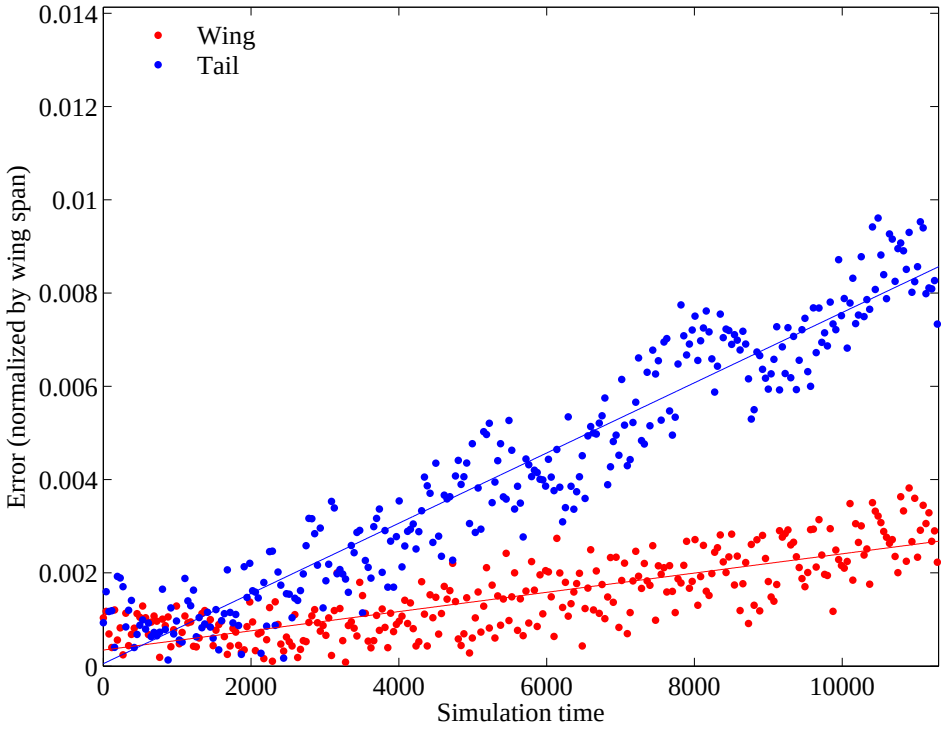


(a) Normalized Kinetic Energy

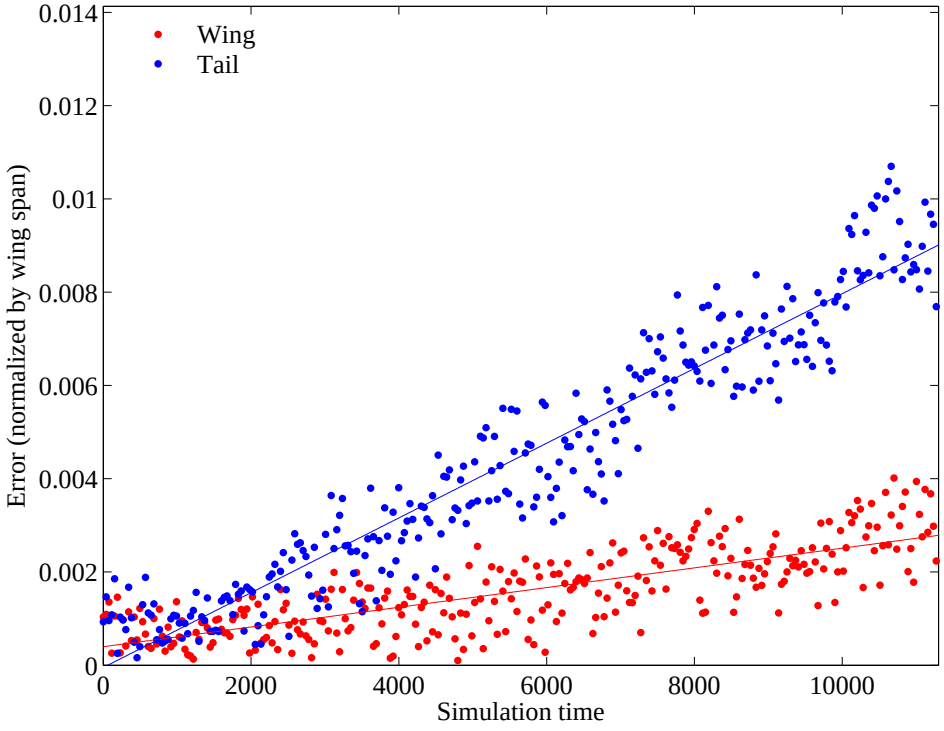


(b) Kinetic Energy Error

Figure 5.12: KE error: Small vs. Medium domain (Degree 9, Coarse grid)

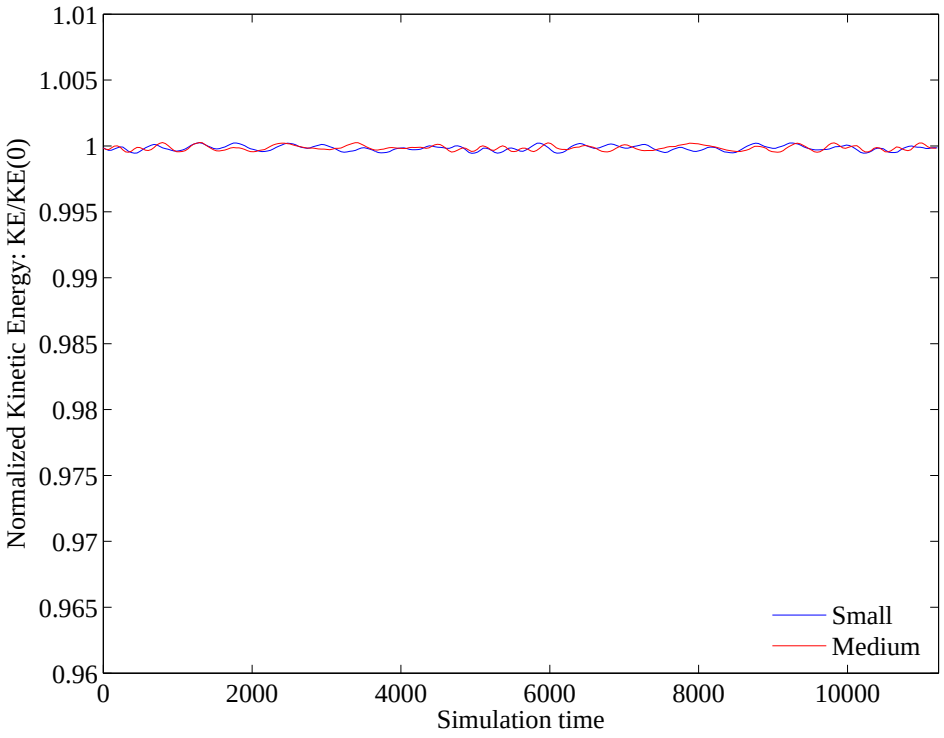


(a) d14 h20

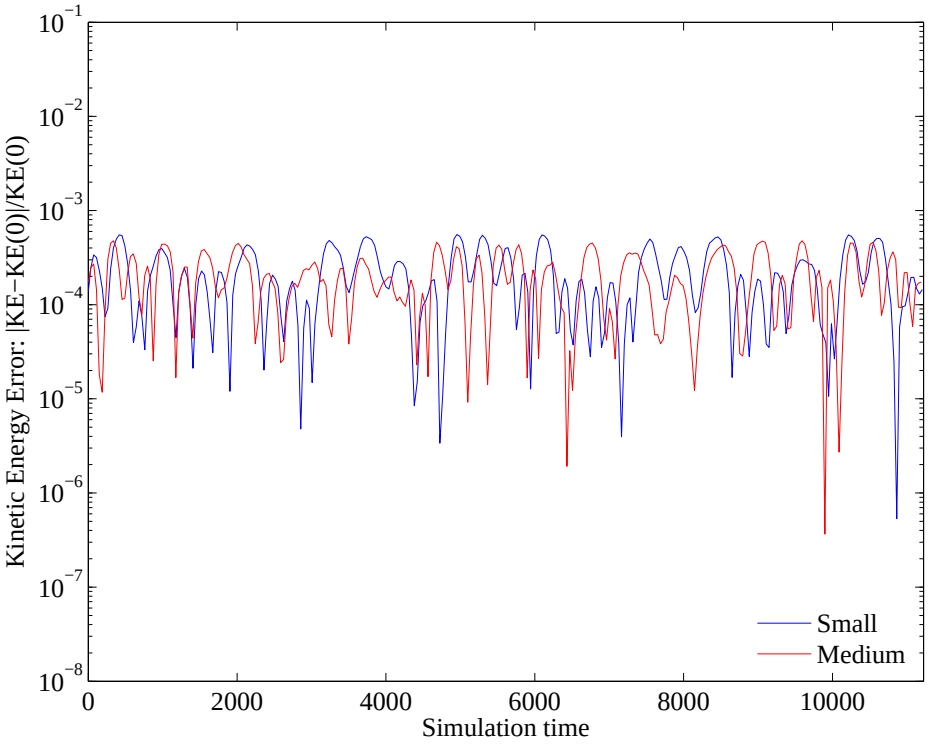


(b) d14 h40

Figure 5.13: Trajectory error: Small vs. Medium domain (Degree 14, Coarse grid)

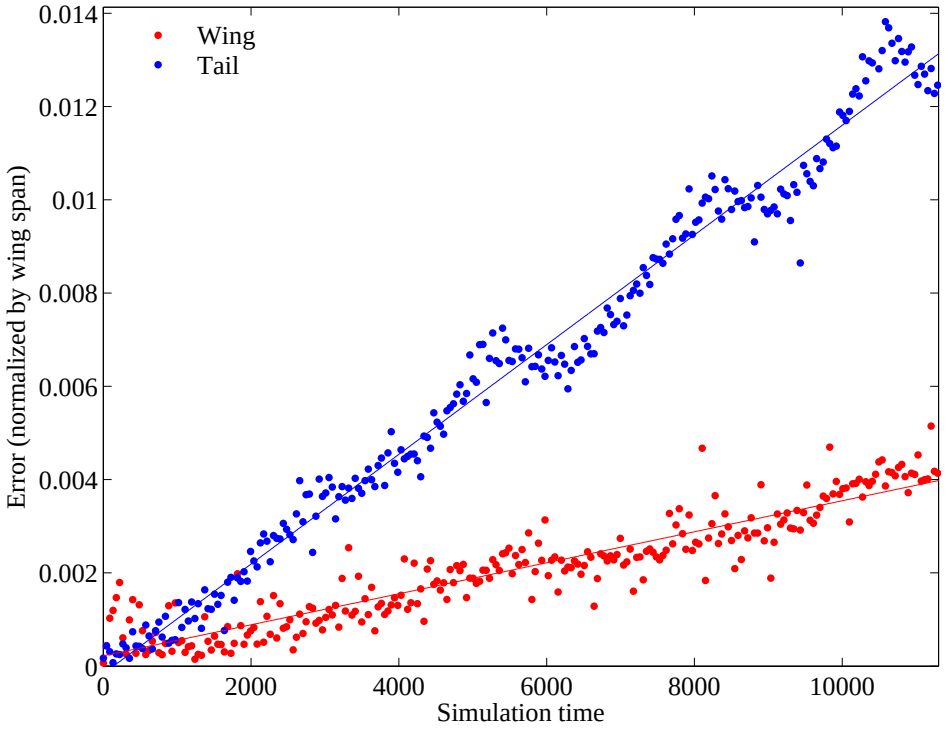


(a) Normalized Kinetic Energy

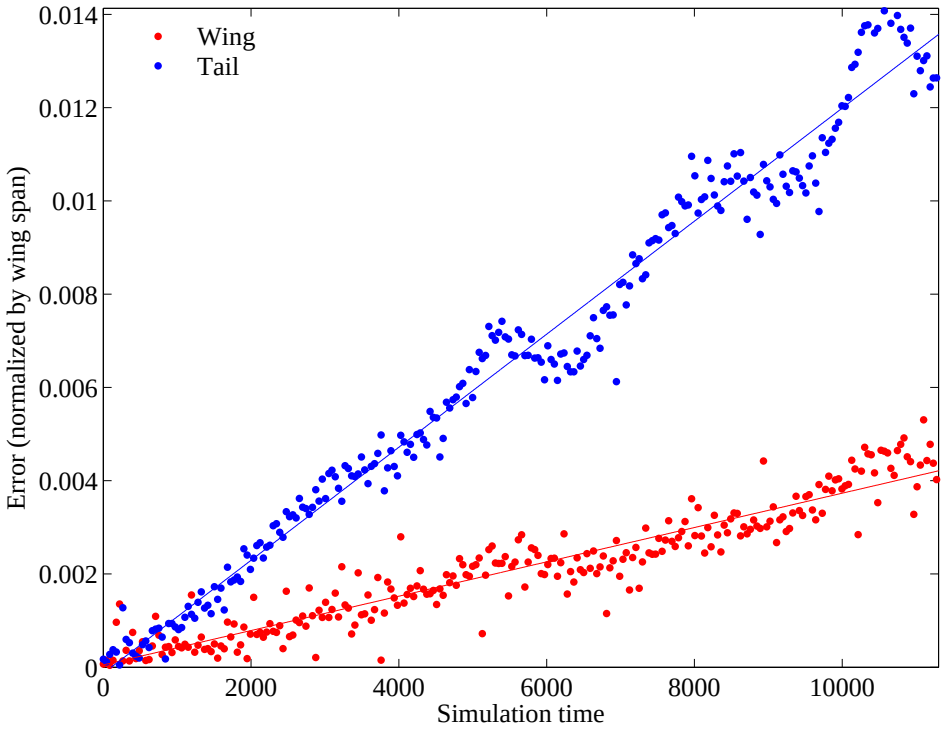


(b) Kinetic Energy Error

Figure 5.14: KE error: Small vs. Medium domain (Degree 14, Coarse grid)

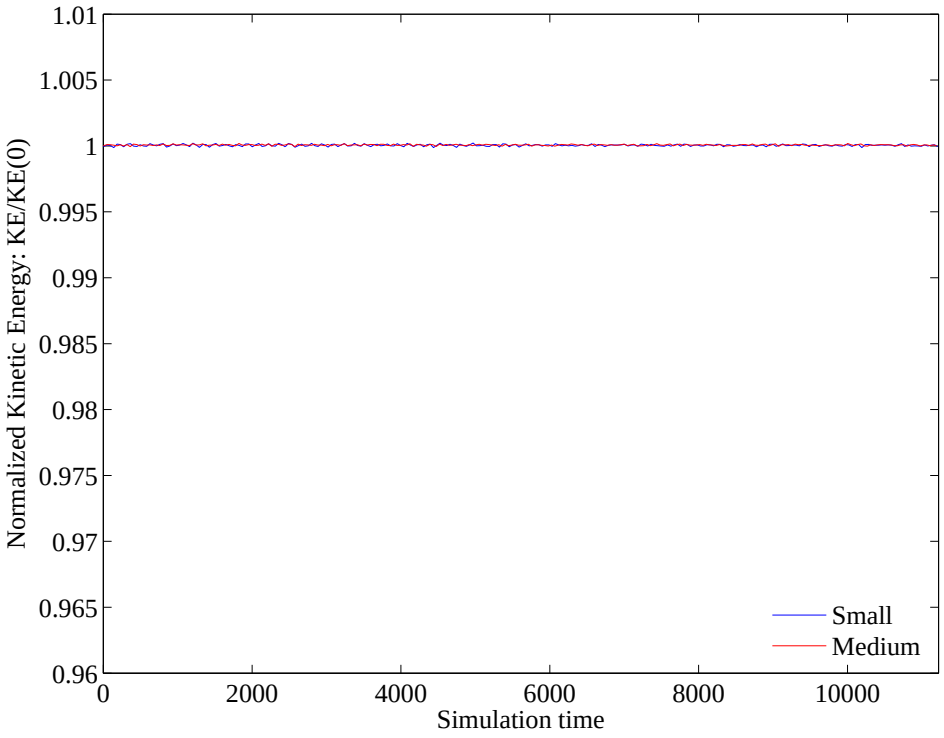


(a) d03 h150

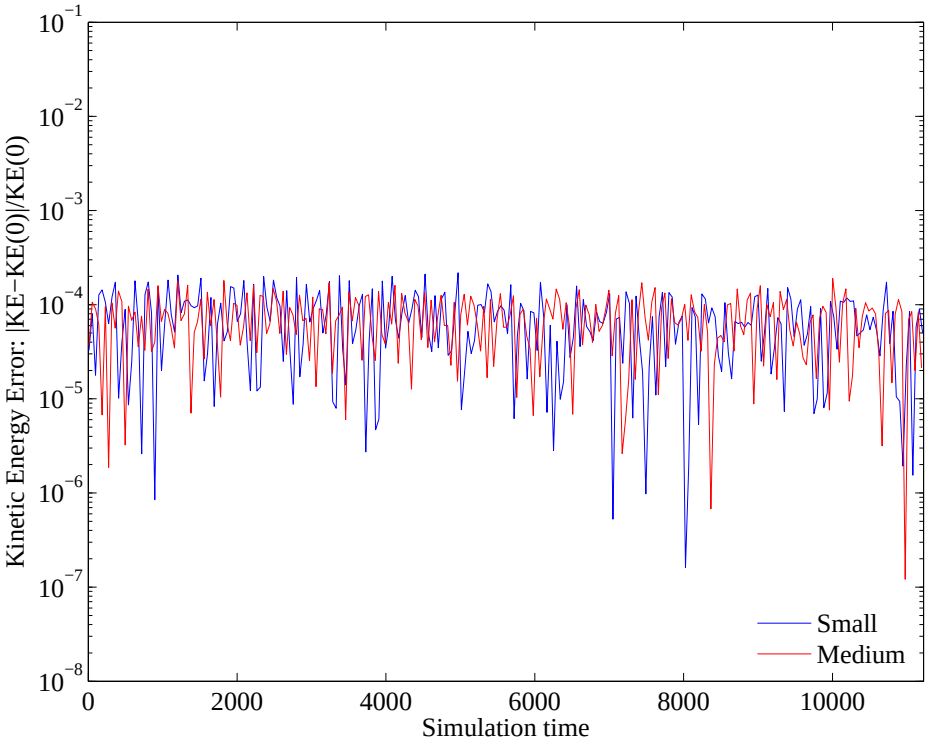


(b) d03 h300

Figure 5.15: Trajectory error: Small vs. Medium domain (Degree 3, Fine grid)

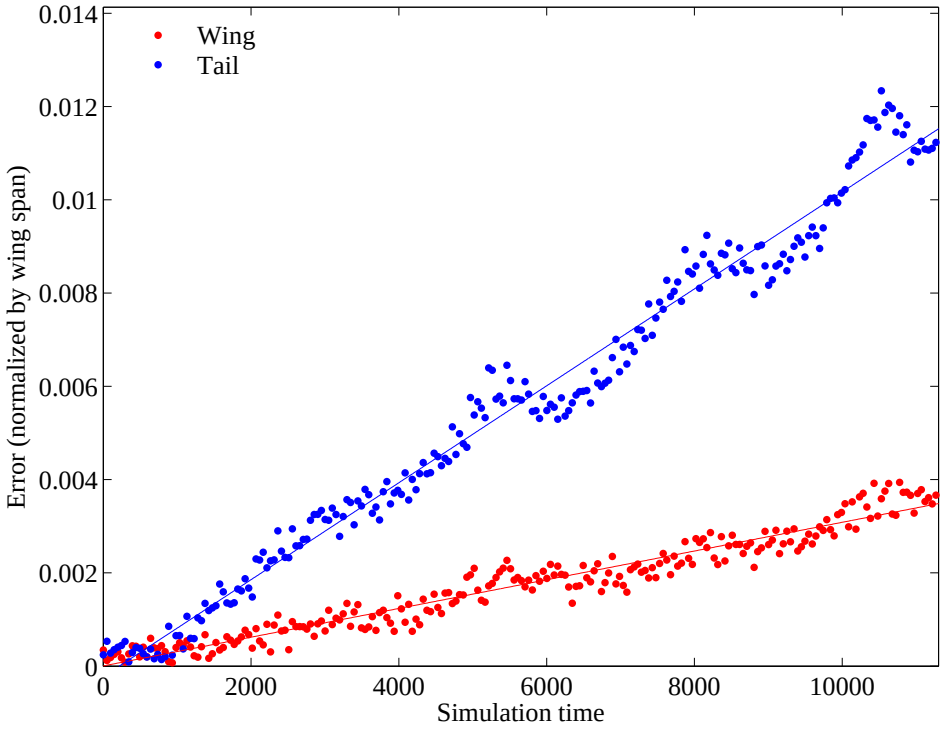


(a) Normalized Kinetic Energy

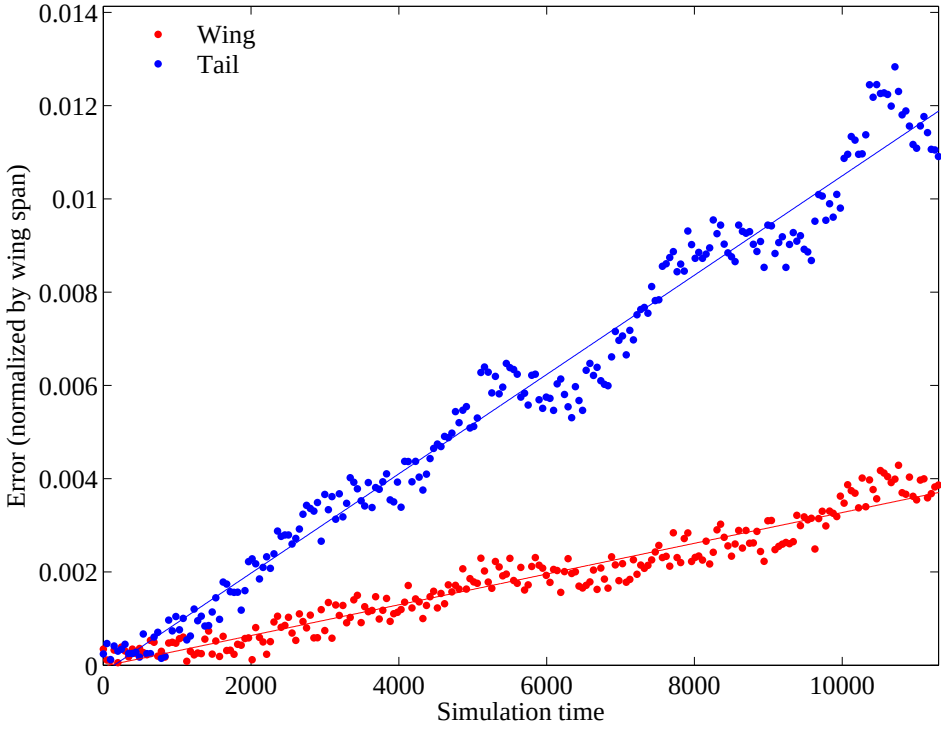


(b) Kinetic Energy Error

Figure 5.16: KE error: Small vs. Medium domain (Degree 3, Fine grid)

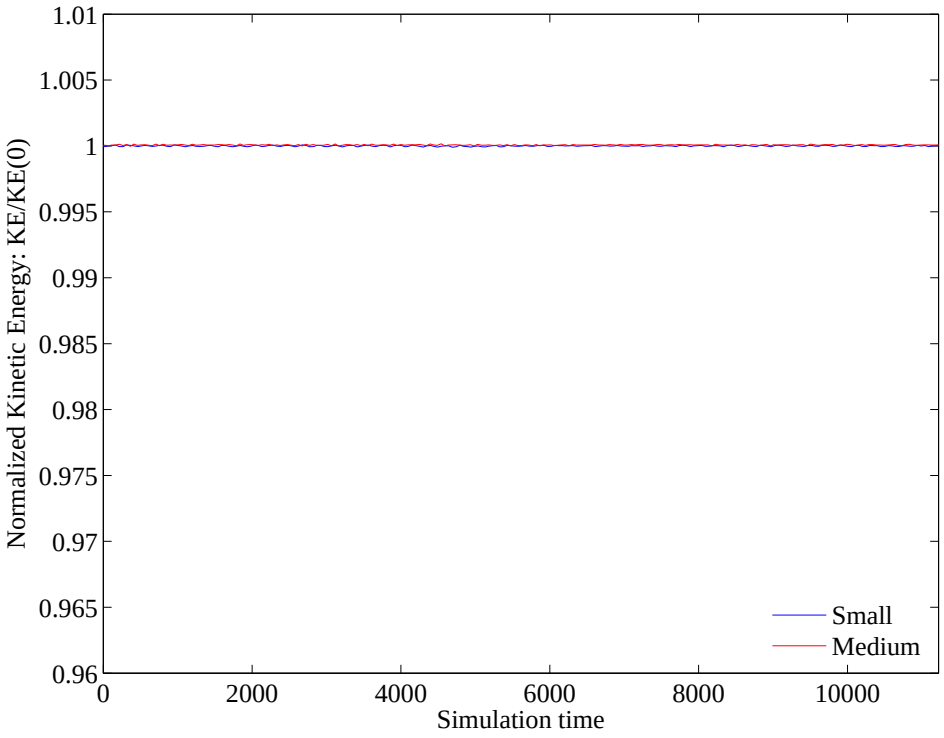


(a) d09 h60

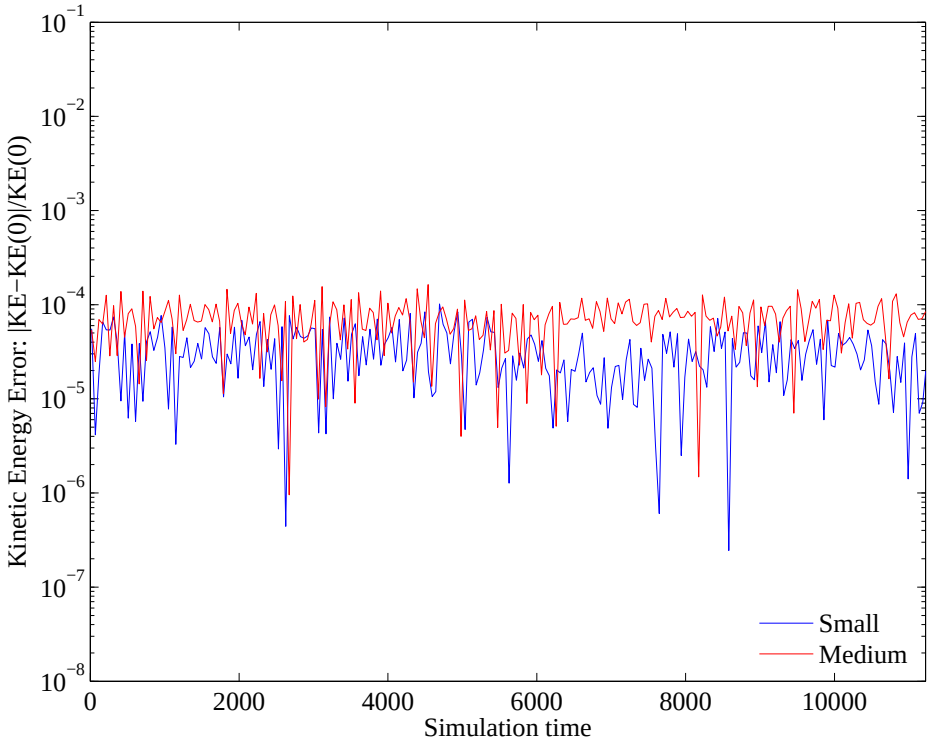


(b) d09 h120

Figure 5.17: Trajectory error: Small vs. Medium domain (Degree 9, Fine grid)

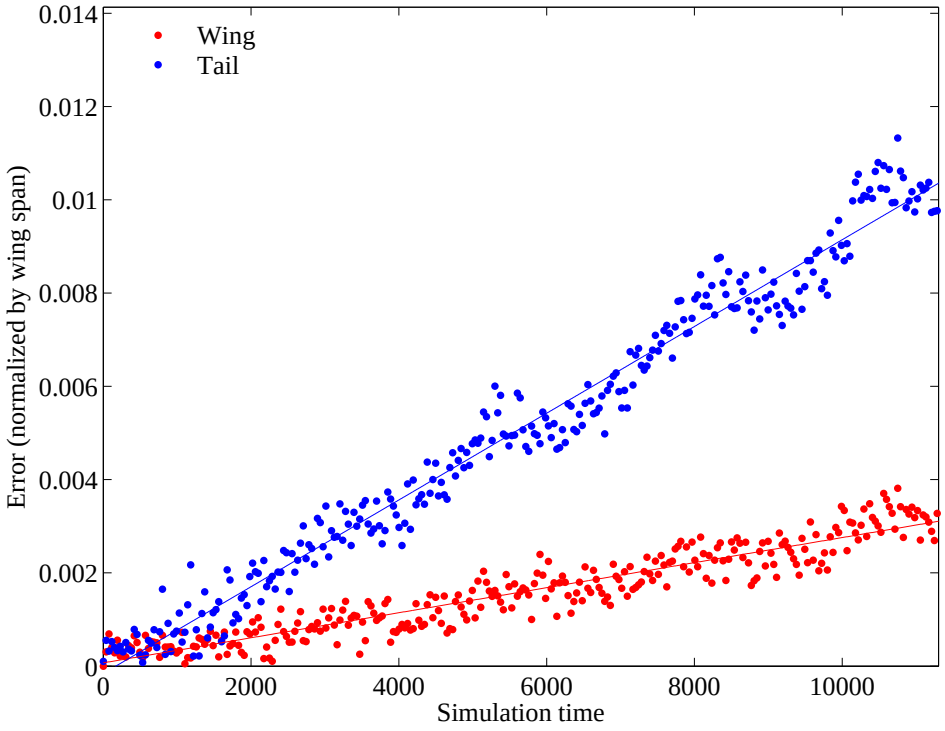


(a) Normalized Kinetic Energy

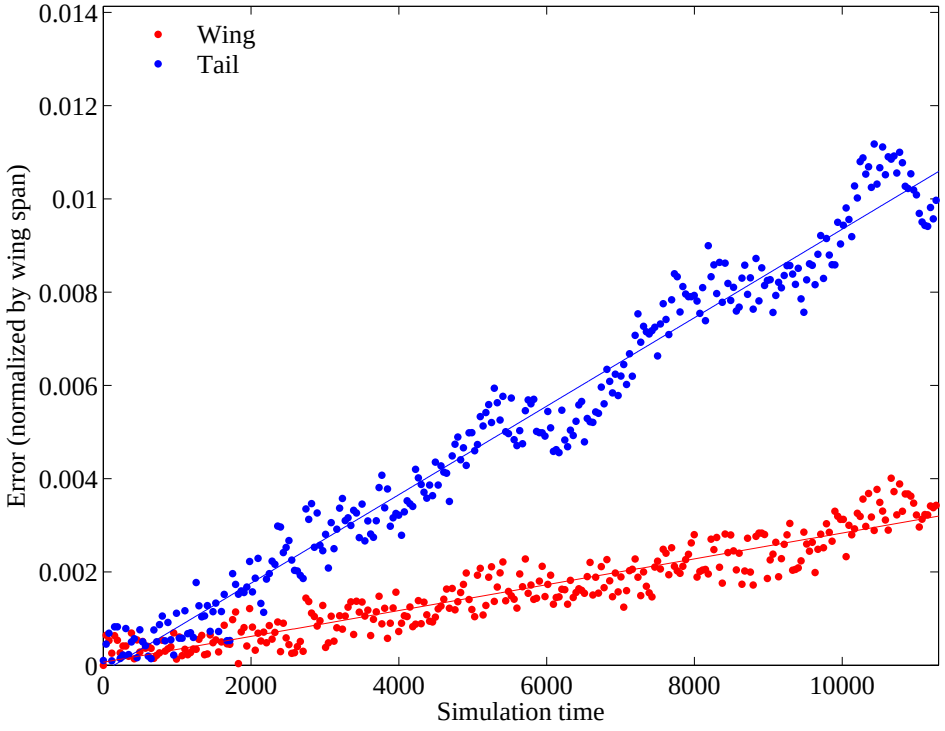


(b) Kinetic Energy Error

Figure 5.18: KE error: Small vs. Medium domain (Degree 9, Fine grid)

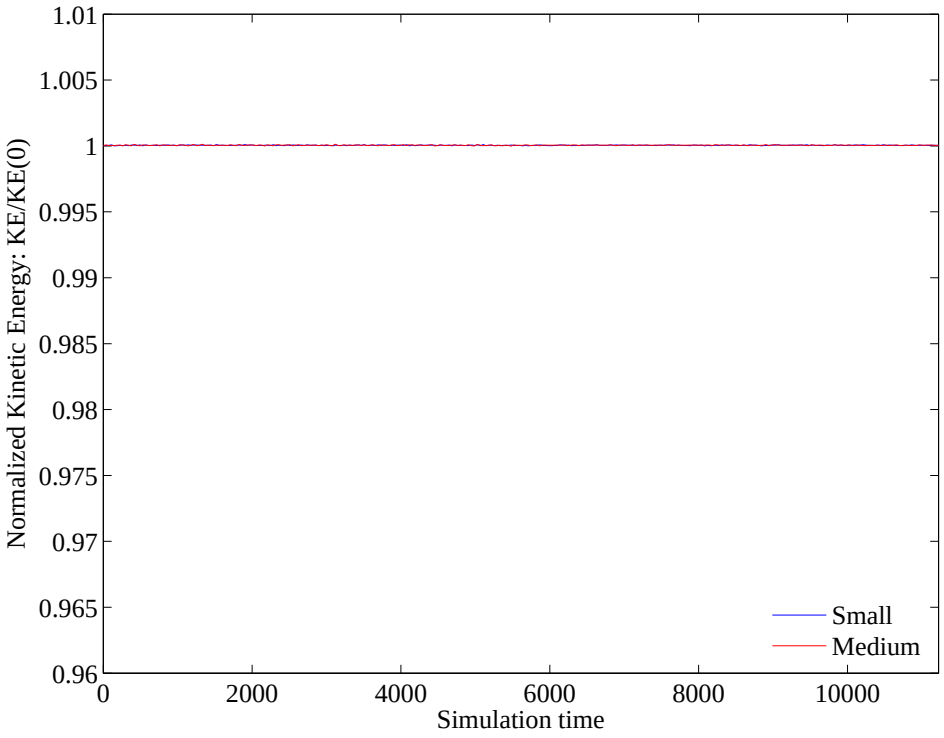


(a) d14 h40

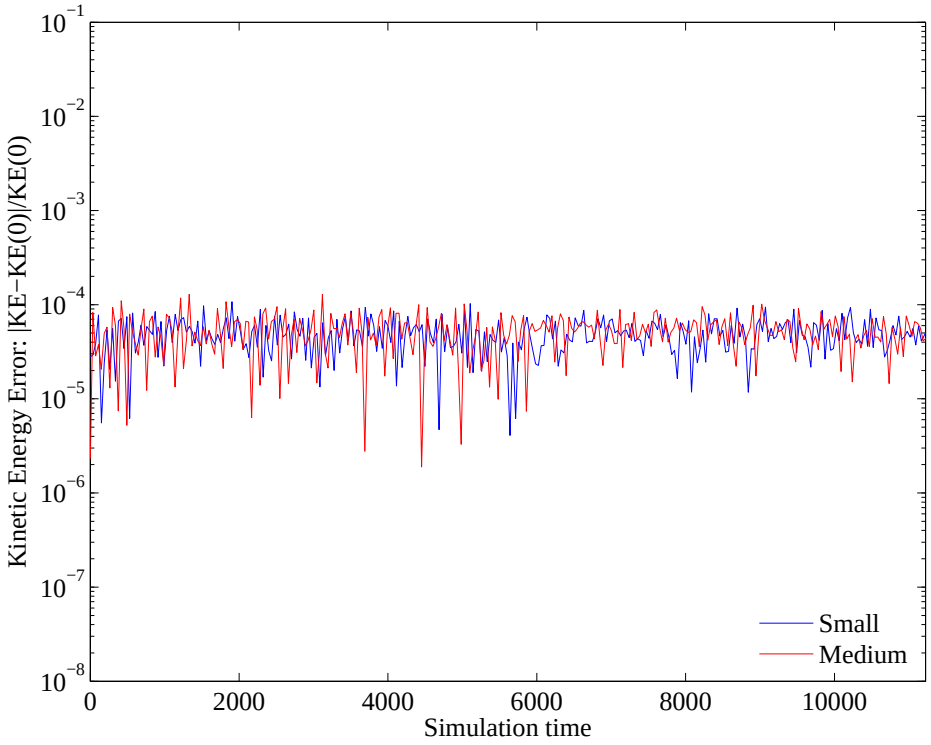


(b) d14 h80

Figure 5.19: Trajectory error: Small vs. Medium domain (Degree 14, Fine grid)



(a) Normalized Kinetic Energy



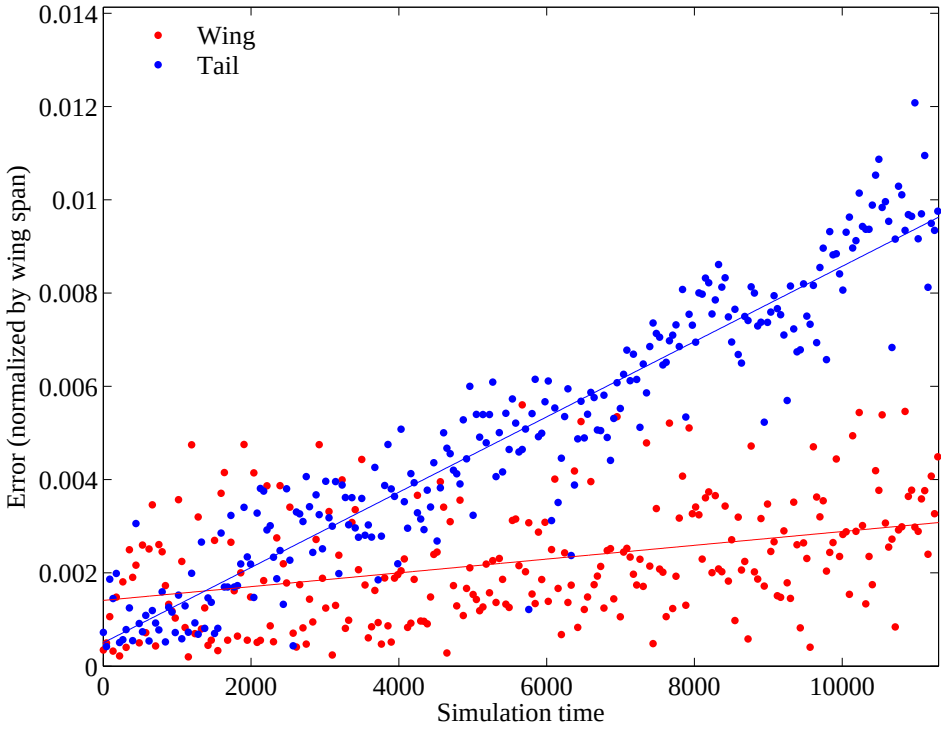
(b) Kinetic Energy Error

Figure 5.20: KE error: Small vs. Medium domain (Degree 14, Fine grid)

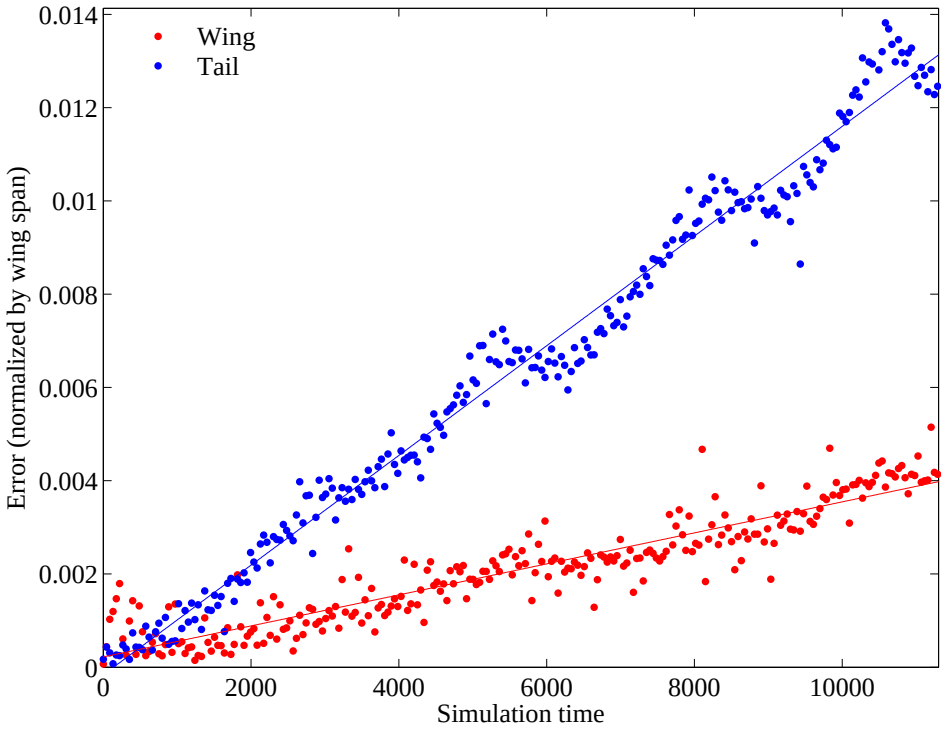
5.3 Cell Density

Cell density effects on the DG method are analyzed by comparing two cell densities denoted as coarse and fine. The data in Figures 5.21 through 5.32 shows that the most noticeable difference between the two densities is that the trajectory error data points are much more focused for the fine density cases. For example, whereas the points seem spread out and more linear in Figure 5.21 (a), a clear oscillatory pattern emerges in the fine case (c) that corresponds to the vortex trajectories. The tradeoff, however, is that although the data is more focused, the overall error and curve fit show slightly higher error for the fine cases. This suggests that the DG method actually performs better when its ability to use fewer cells with higher polynomials is exploited.

The effect of cell density on kinetic energy seems to be the opposite. The normalized values are more oscillatory for the coarse cases, and sometimes even lower (as in the degree 3 cases of Figures 5.22 and 5.28). Also, the kinetic energy error plots consistently show higher KE error for the coarse cases, with the increase anywhere from roughly half an order of magnitude to a little over one order of magnitude. Thus, a coarser cell density will produce lower trajectory error, but slightly higher kinetic energy error.

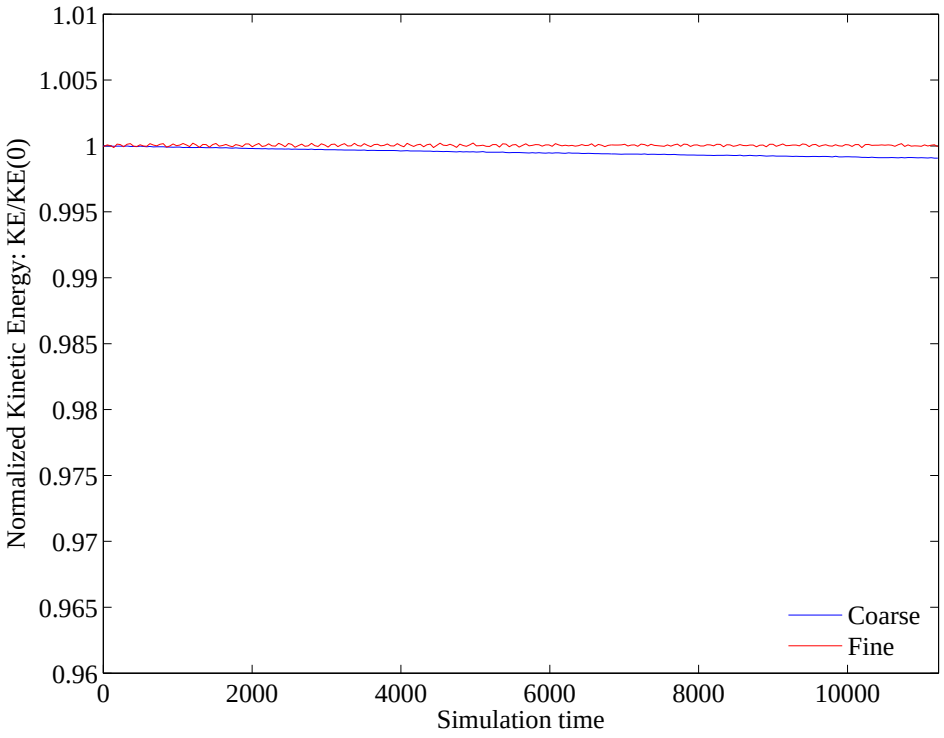


(a) d03 h75

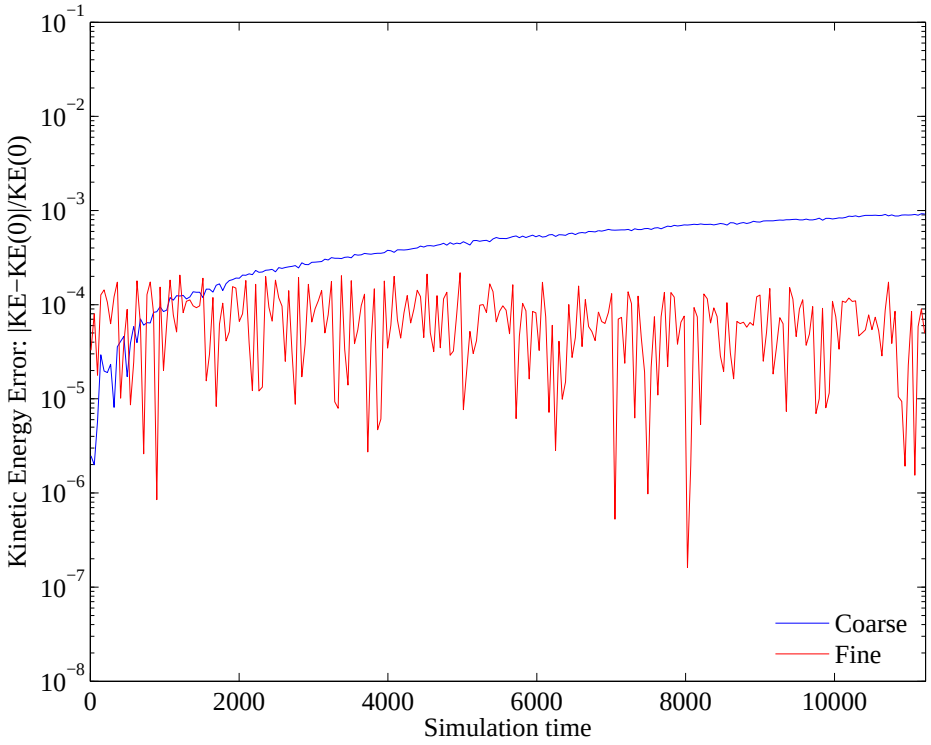


(b) d03 h150

Figure 5.21: Trajectory error: Coarse vs. Fine density (Small domain, Degree 3)

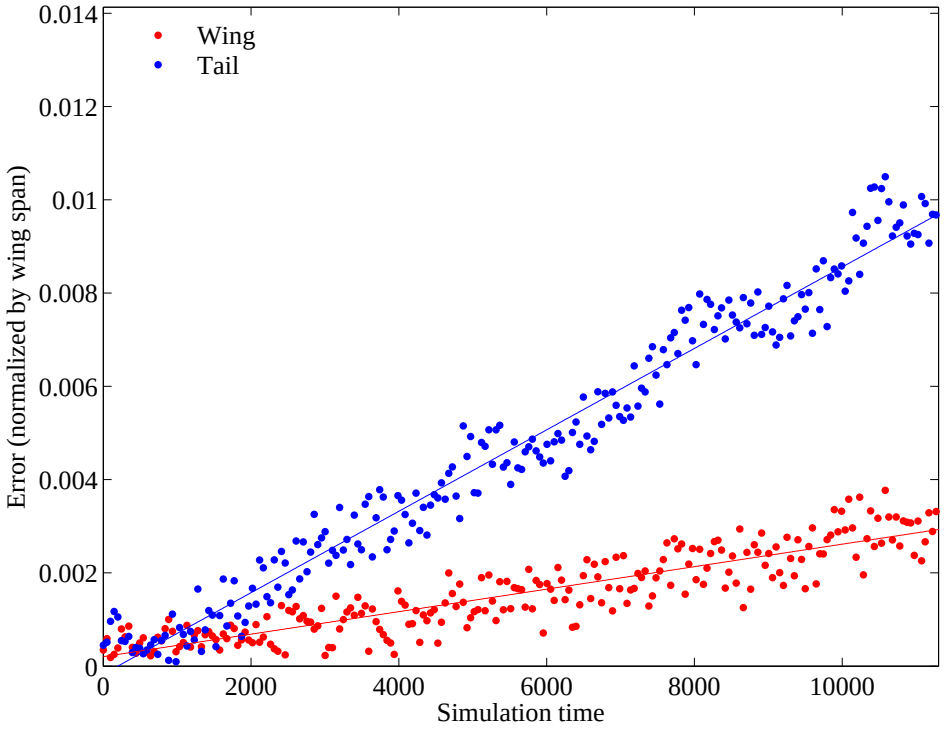


(a) Normalized Kinetic Energy

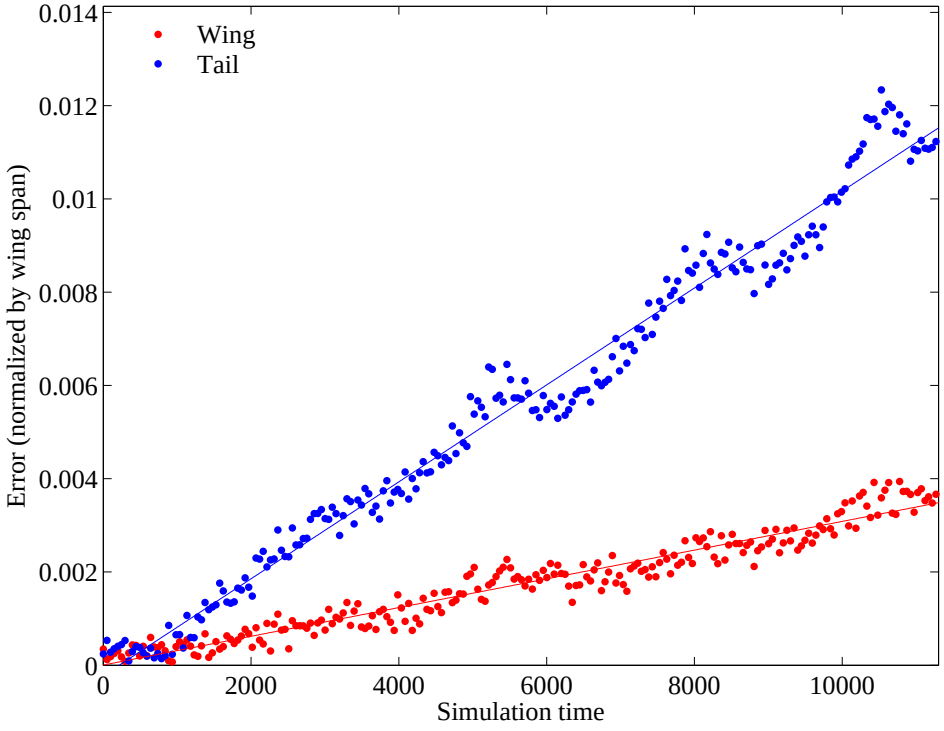


(b) Kinetic Energy Error

Figure 5.22: KE error: Coarse vs. Fine density (Small domain, Degree 3)

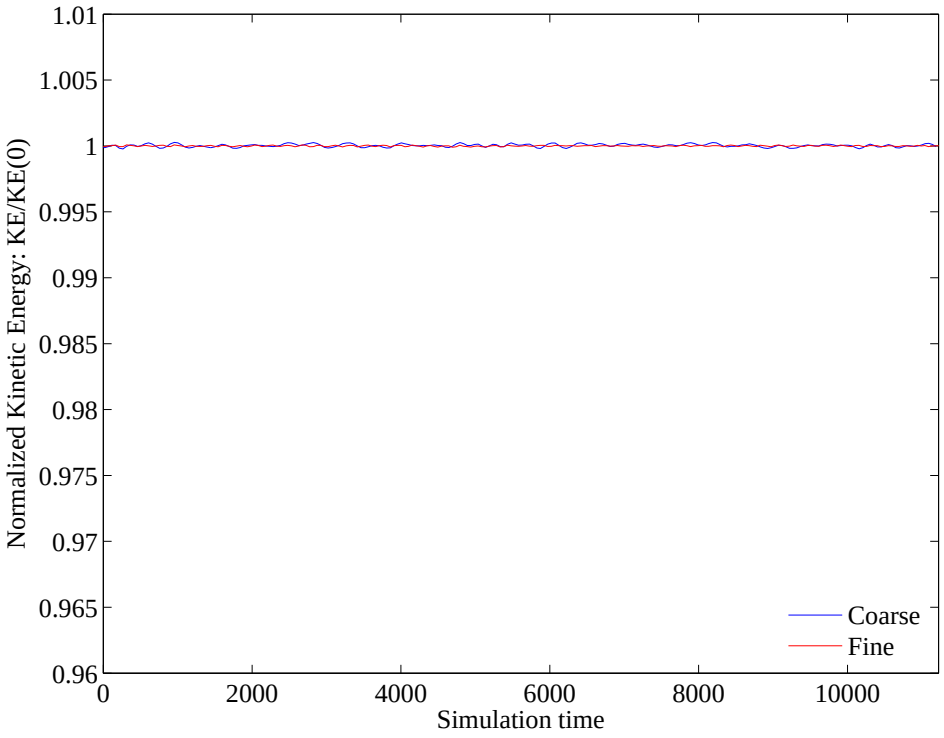


(a) d09 h30

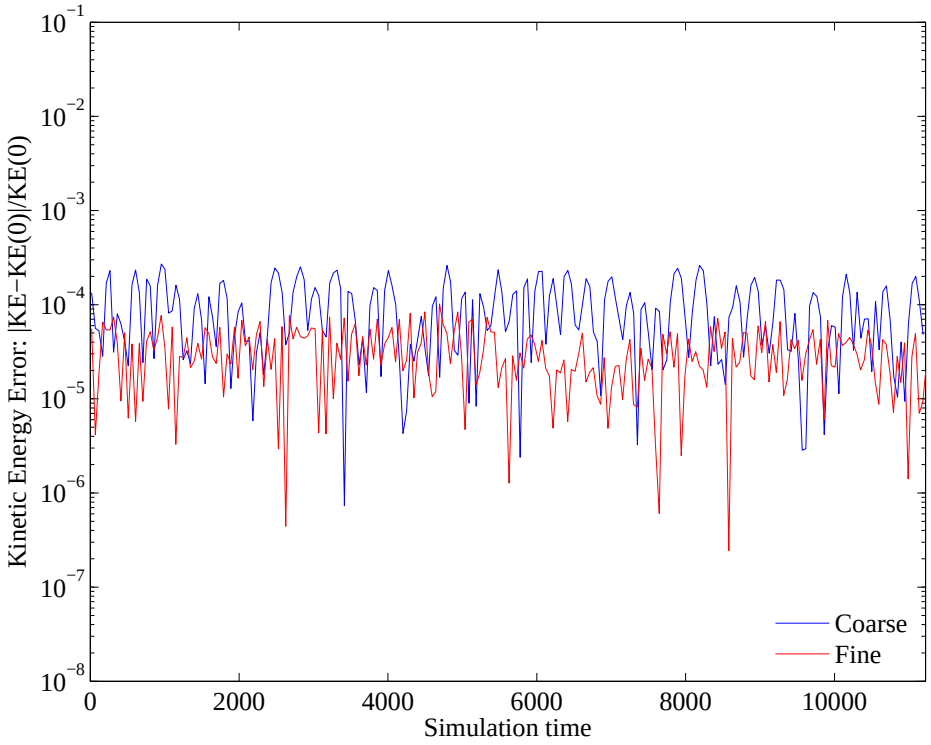


(b) d09 h60

Figure 5.23: Trajectory error: Coarse vs. Fine density (Small domain, Degree 9)

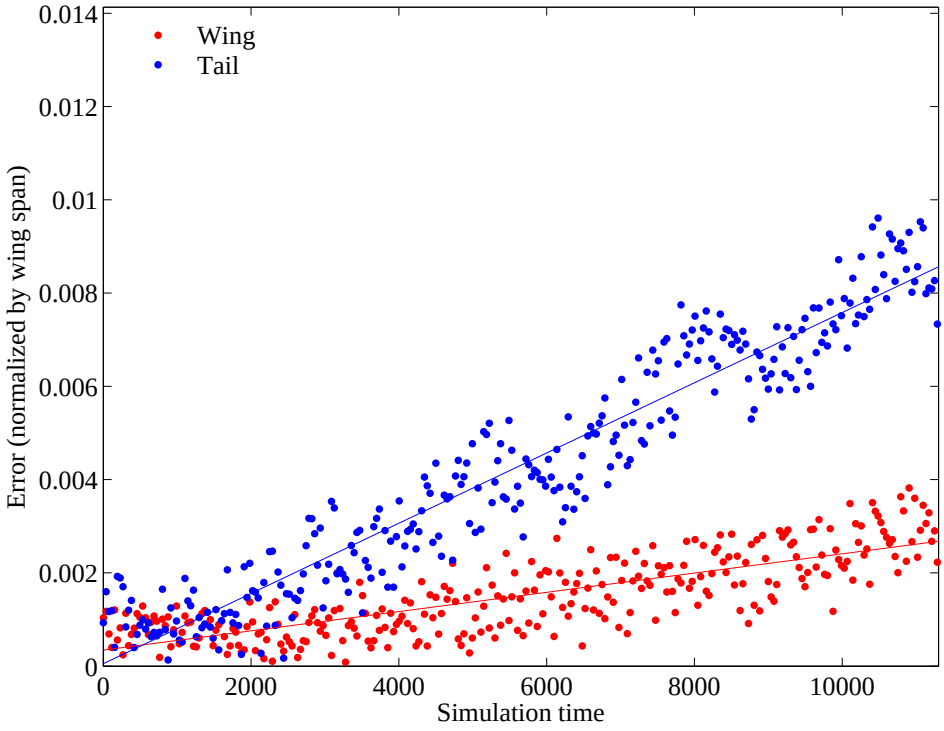


(a) Normalized Kinetic Energy

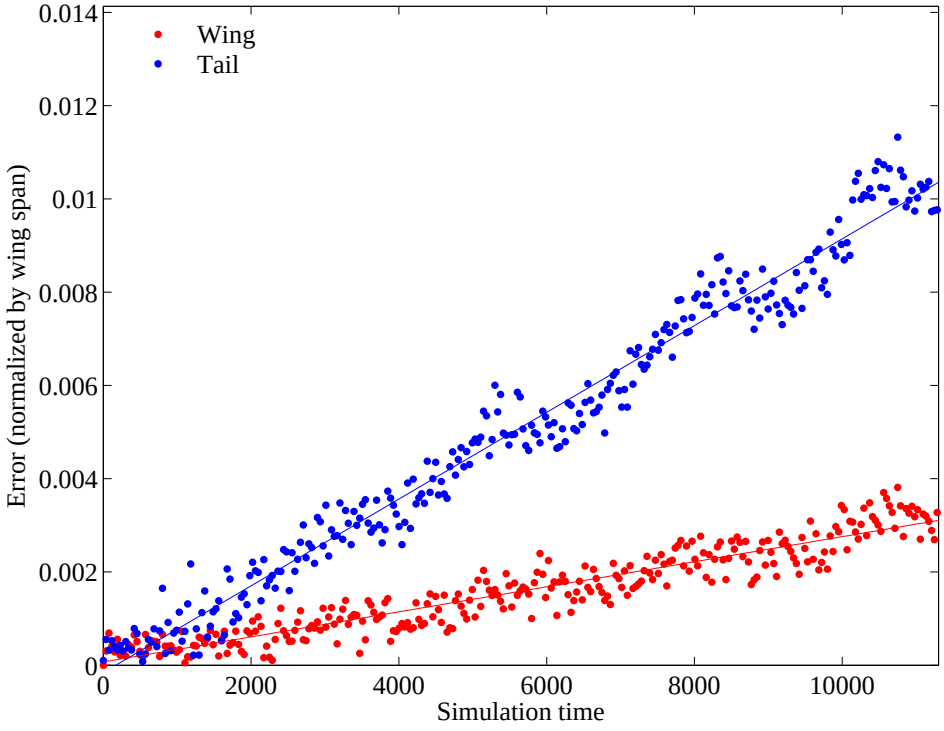


(b) Kinetic Energy Error

Figure 5.24: KE error: Coarse vs. Fine density (Small domain, Degree 9)

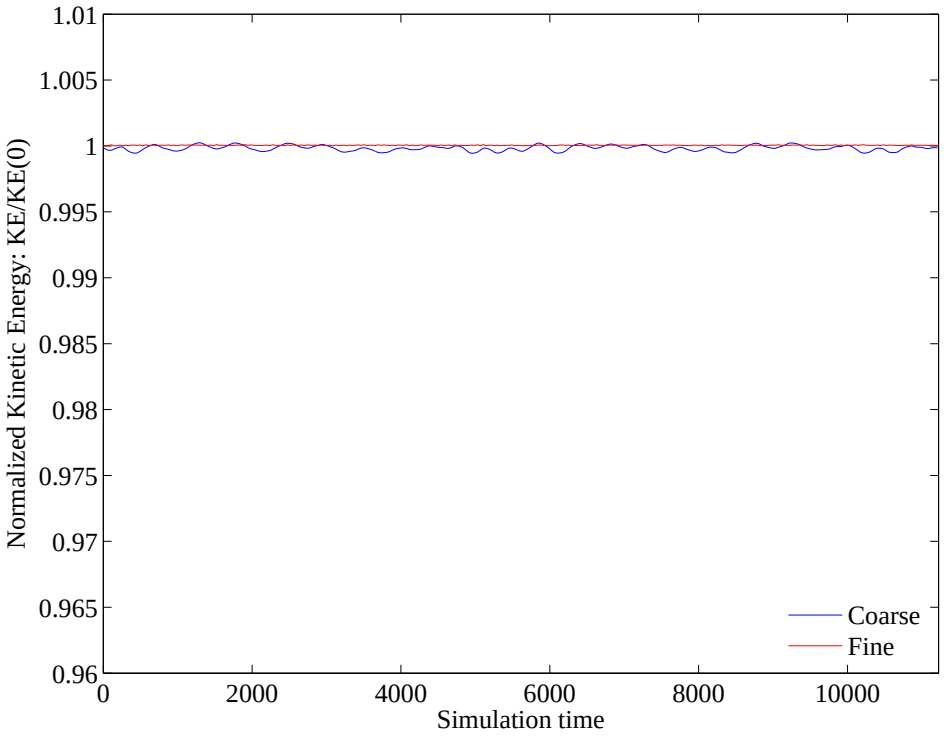


(a) d14 h20

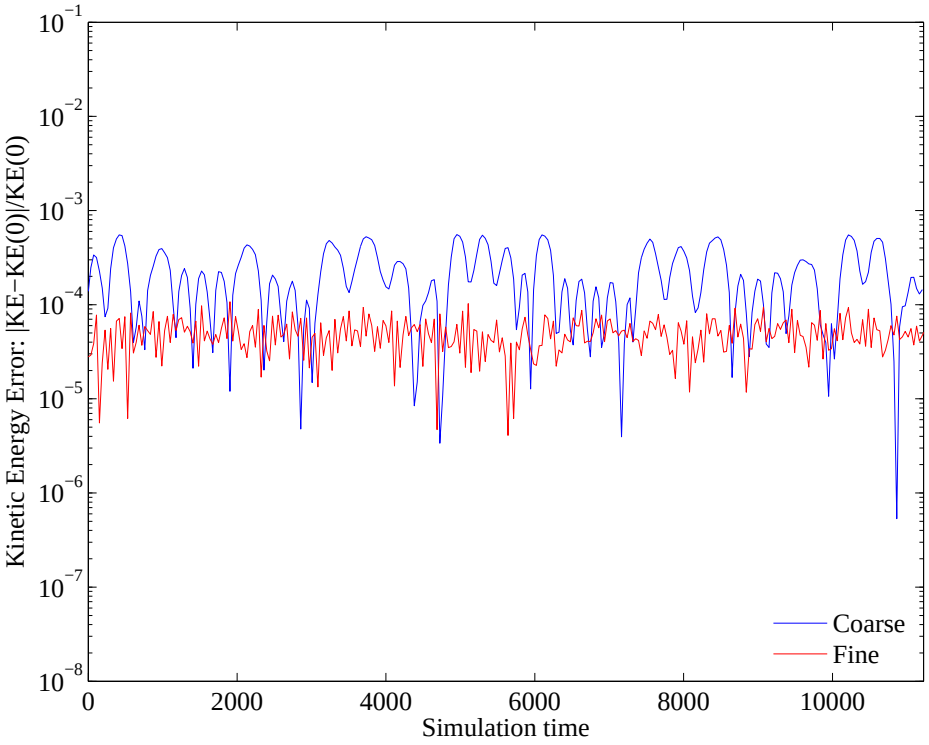


(b) d14 h40

Figure 5.25: Trajectory error: Coarse vs. Fine density (Small domain, Degree 14)

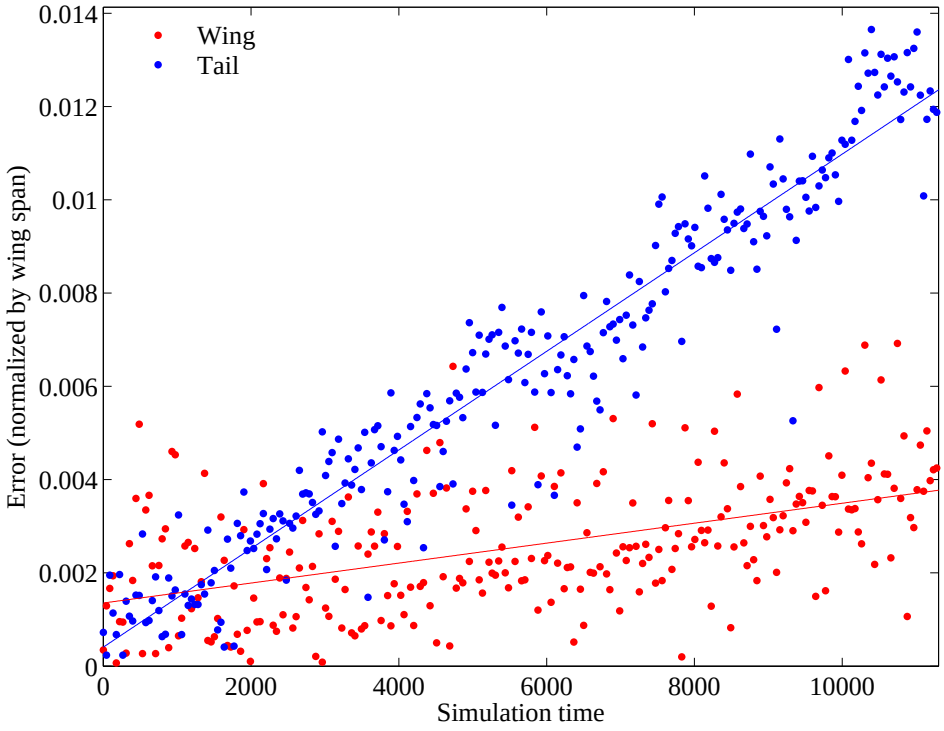


(a) Normalized Kinetic Energy

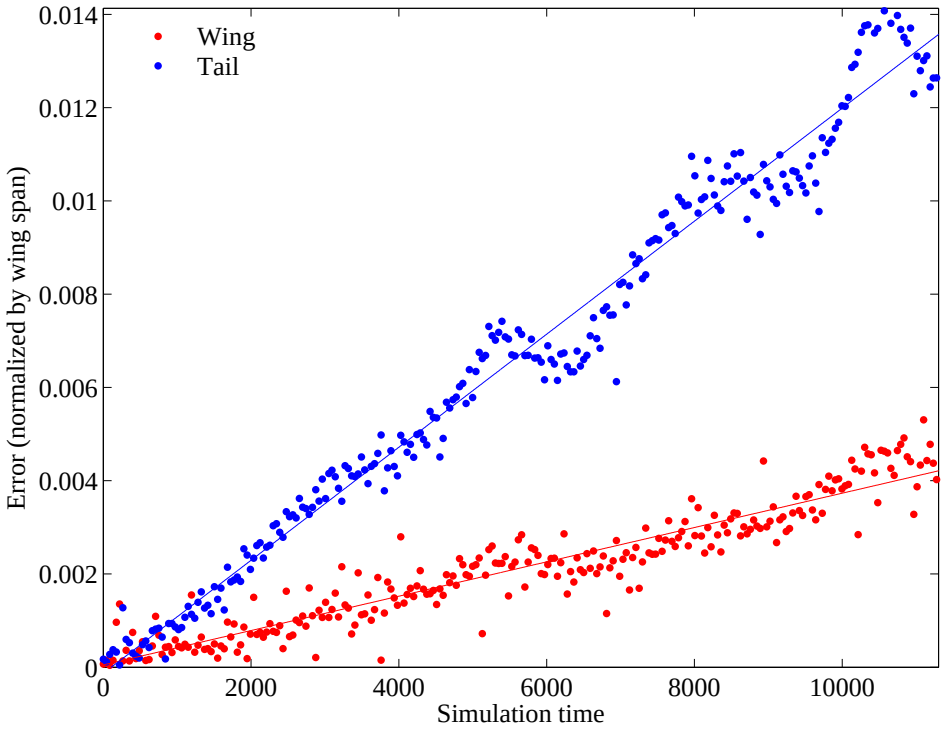


(b) Kinetic Energy Error

Figure 5.26: KE error: Coarse vs. Fine density (Small domain, Degree 14)

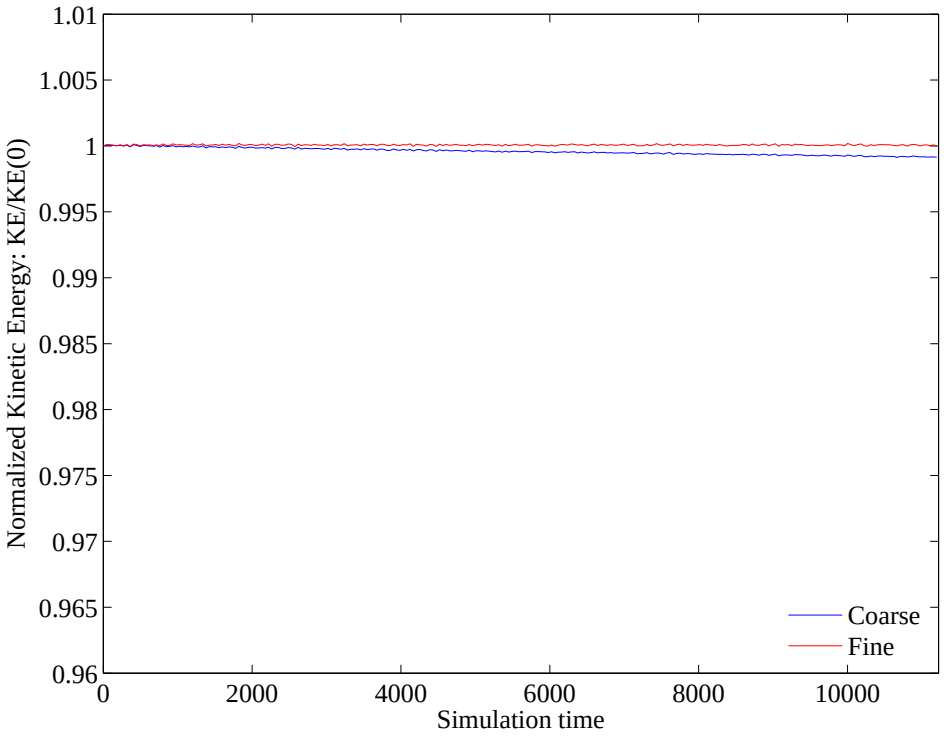


(a) d03 h150

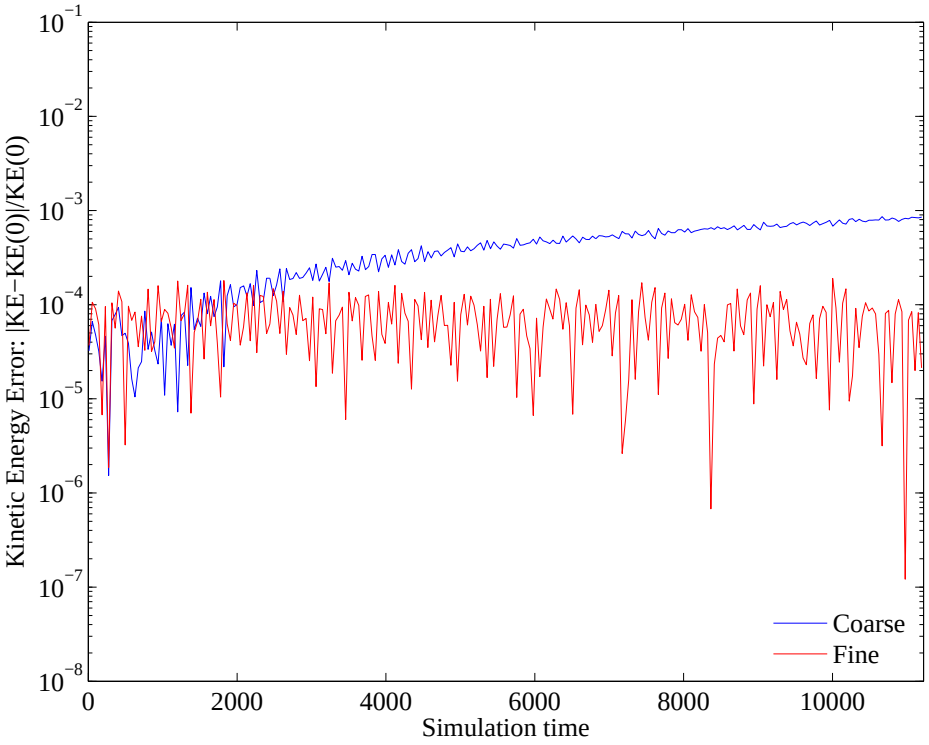


(b) d03 h300

Figure 5.27: Trajectory error: Coarse vs. Fine density (Medium domain, Degree 3)

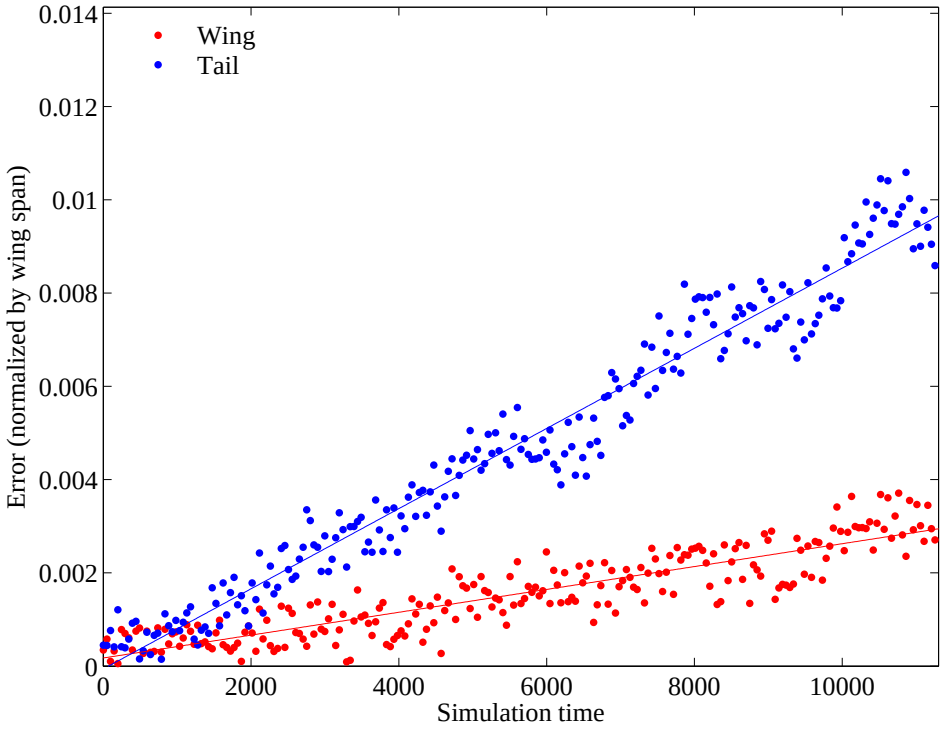


(a) Normalized Kinetic Energy

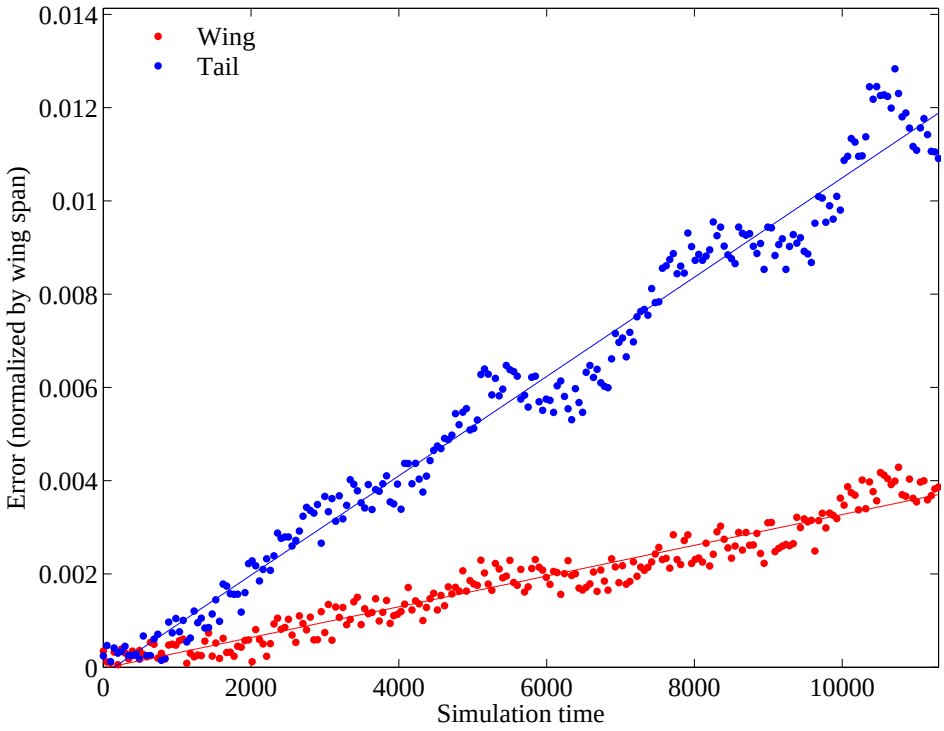


(b) Kinetic Energy Error

Figure 5.28: KE error: Coarse vs. Fine density (Medium domain, Degree 3)

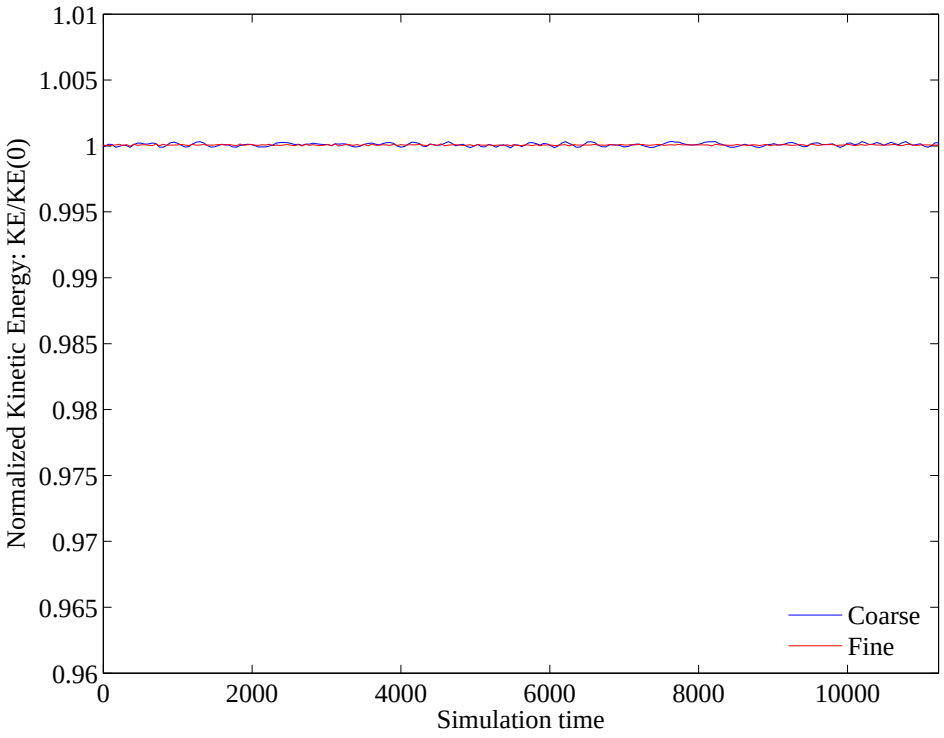


(a) d09 h60

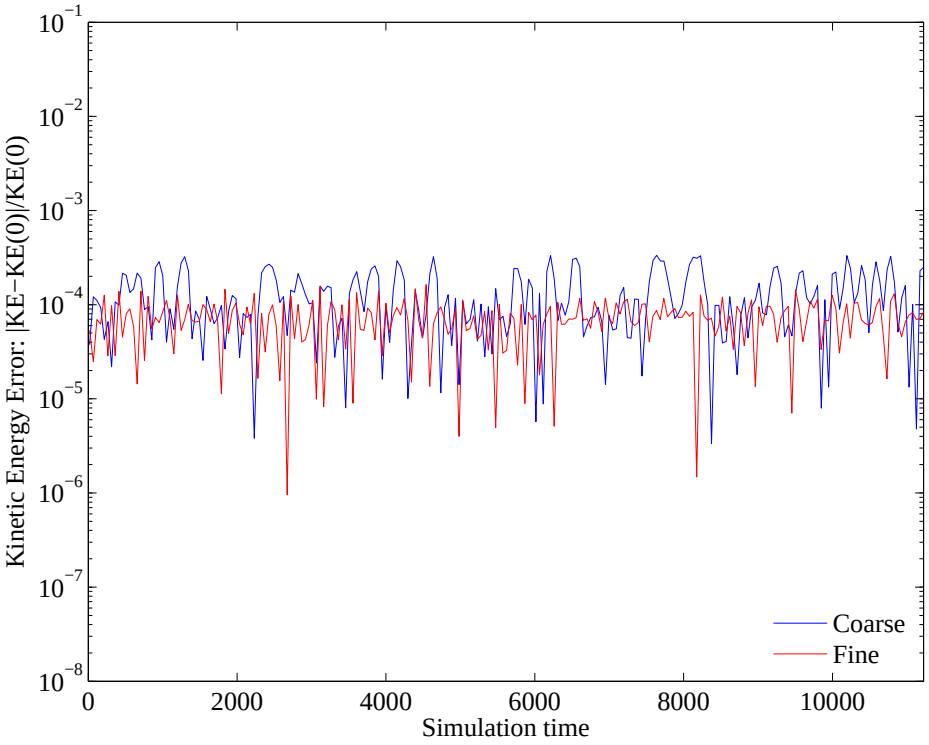


(b) d09 h120

Figure 5.29: Trajectory error: Coarse vs. Fine density (Medium domain, Degree 9)

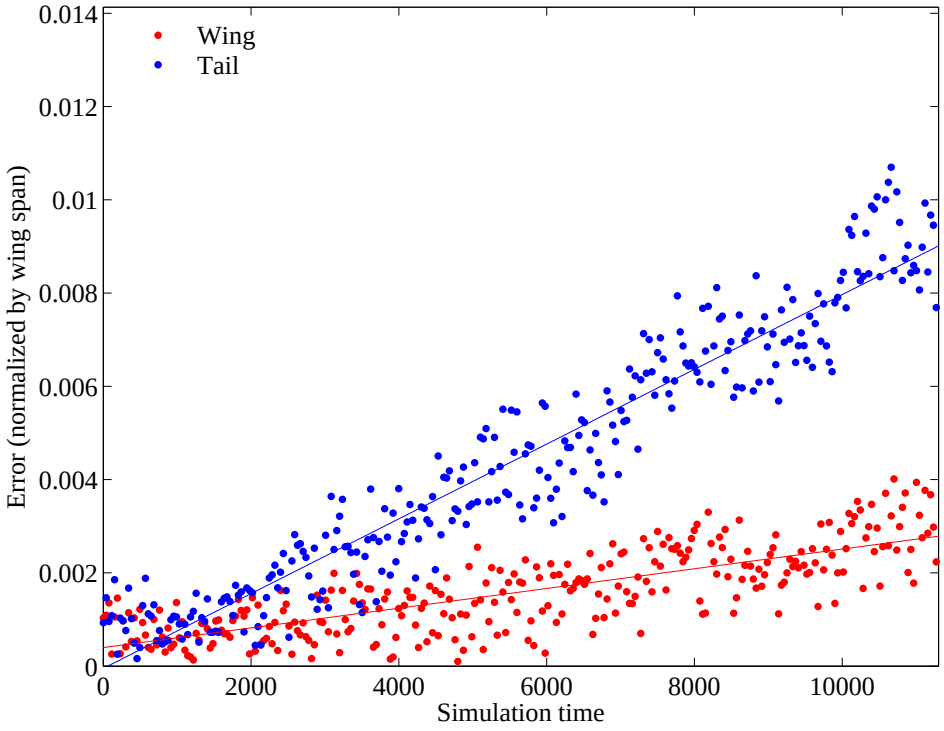


(a) Normalized Kinetic Energy

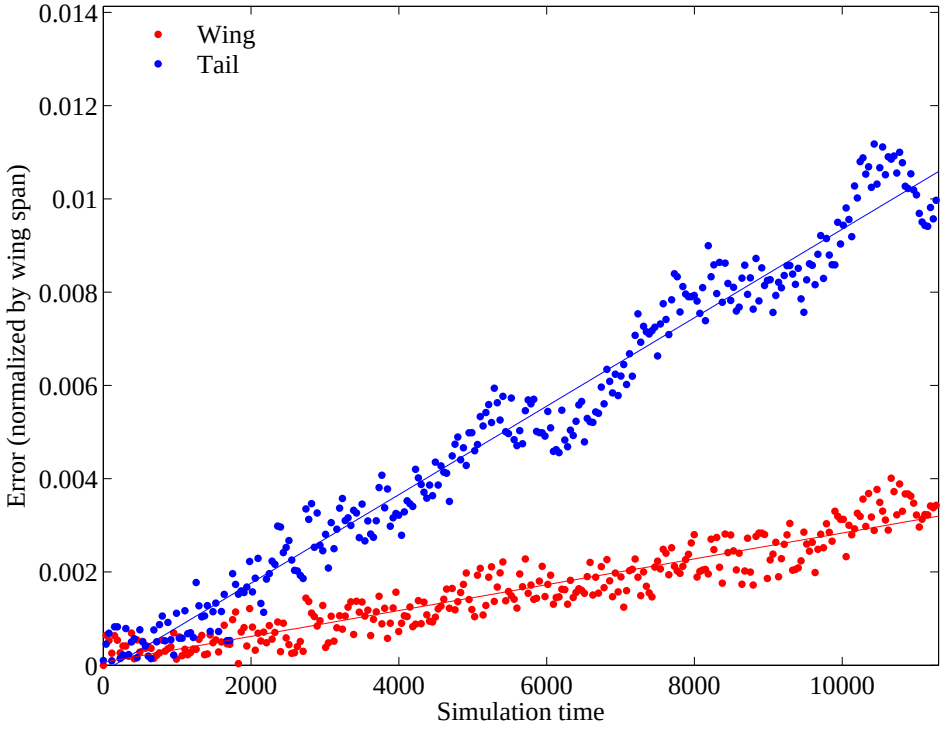


(b) Kinetic Energy Error

Figure 5.30: KE error: Coarse vs. Fine density (Medium domain, Degree 9)

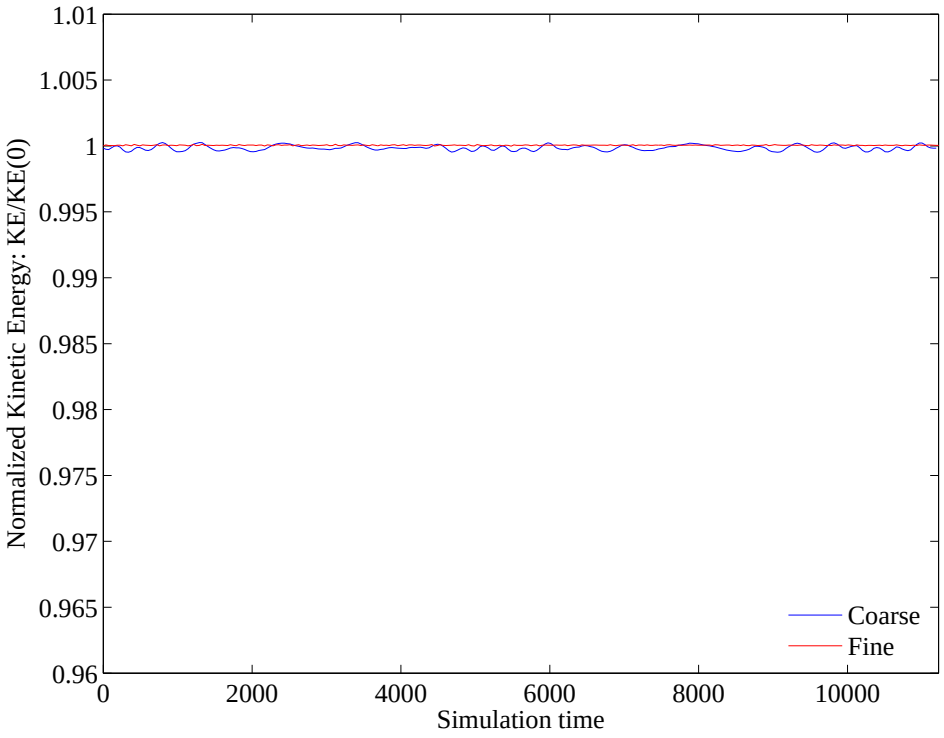


(a) d14 h40

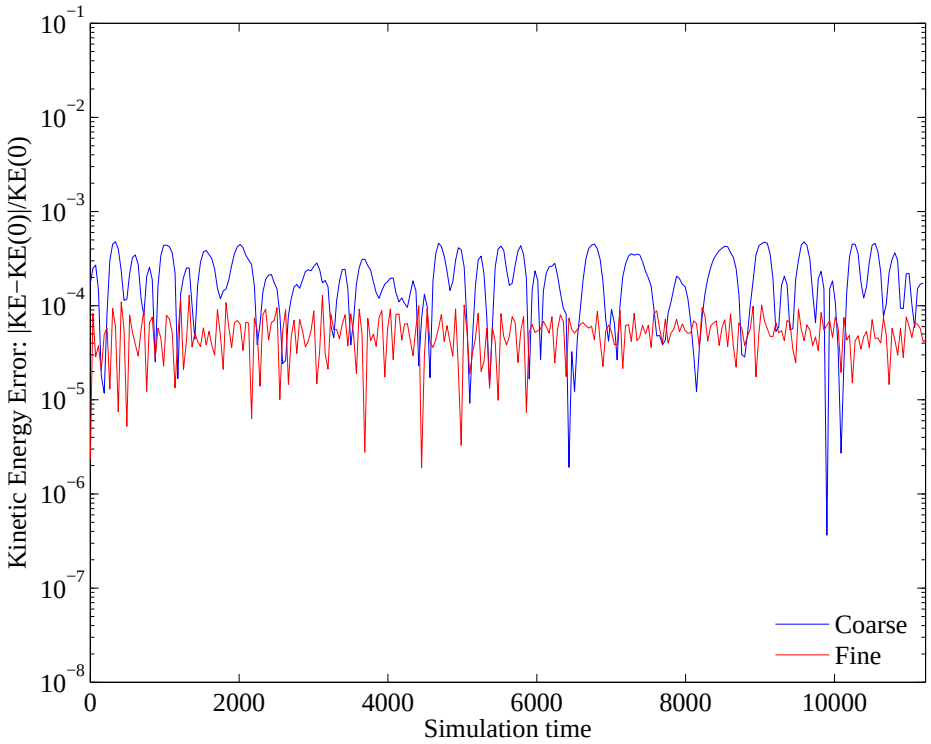


(b) d14 h80

Figure 5.31: Trajectory error: Coarse vs. Fine density (Medium domain, Degree 14)



(a) Normalized Kinetic Energy



(b) Kinetic Energy Error

Figure 5.32: KE error: Coarse vs. Fine density (Medium domain, Degree 14)

5.4 Further Investigation

5.4.1 Large domain size

Since the preliminary cases outlined above were only intended as a starting point, it is unsurprising that the results from those cases suggested the need for additional cases. The first parameter that seems to warrant more exploring is domain size. It is clear from the Lagrangian cases that when comparing to the unbounded case, the medium case is a much better match than the small domain, which severely constrains the vortices. Thus, it is not a stretch to suggest that doubling the domain again from the medium size would be a still better approximation to the unbounded interaction, but the real question is *how much* better. Since doubling the domain size is so expensive computationally (roughly four times as costly), both a coarse and a fine grid for degree 3 are used to save the additional time a degree 9 or 14 case would take. Continuing in the pattern, this means the coarse case uses 300 x 480 cells to model a domain of [0,160] x [-128,128] and the fine case uses 600 x 960 cells.

Before implementing DG on a large domain, Lagrangian methods of Chapter 3 are first used to create an exact result for comparison, and Figure 5.33 reveals that, as expected, the large domain does the best job yet of matching the unbounded vortex trajectory. The limits for the error plot are the same as in Figure 3.9 and it's clear that the error for the large case is much lower even than for the medium case. With a final error of about .05, or 5% of the wingspan, the Large domain seems to be a good tradeoff between computational time and error. More than likely, any further doubling of the domain would increase computational time significantly without a correspondingly vast reduction in error.

With the exact results for comparison, the 'large coarse' and 'large fine' cases are run, and the results are shown in Figures 5.34 and 5.36. When compared with the degree 3 cases from the small and medium domains, the error remains largely unaffected or increases only

minimally, so domain size does not appear to have much effect on the DG method itself where trajectory error is concerned.

The general kinetic energy results from Section 5.2 are expanded and shown to continue with the large domain matching the small and medium domain values closely. In Figure 5.35 (a), the normalized value decreases, but since they all decrease in the same way, the domain size has no effect. It is very likely that a third-degree coarse grid on a domain size double or even quadruple that of the large domain would still exhibit the same trend. The KE error quantifies the similarities in Figure 5.35 (b), where there is a slight decrease in error with the larger domain sizes early on in the simulation, but ultimately converges to the same value. Figure 5.37 (b) displays errors for the fine case that are nearly identical for the entire simulation.

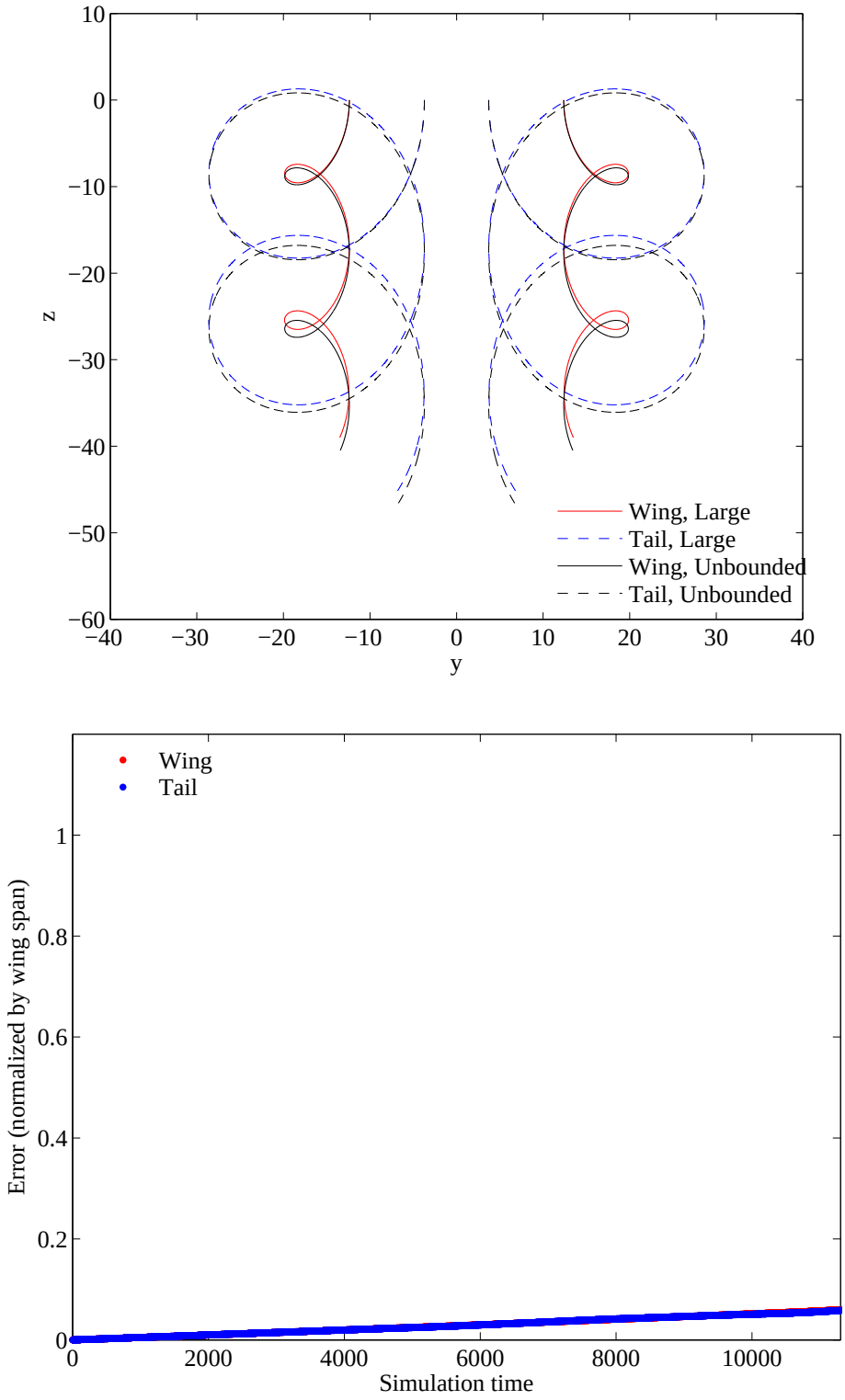
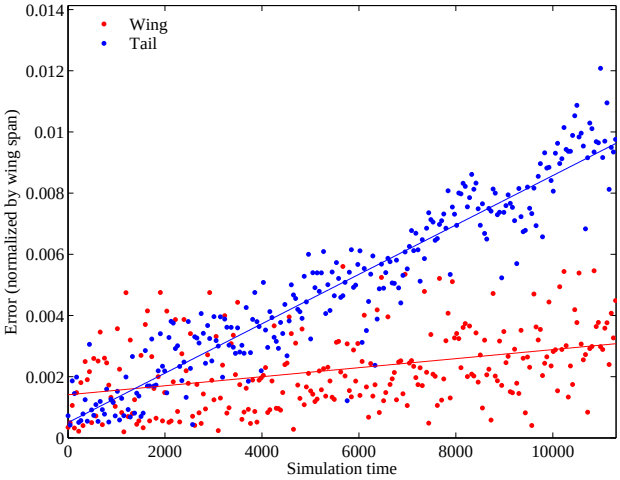
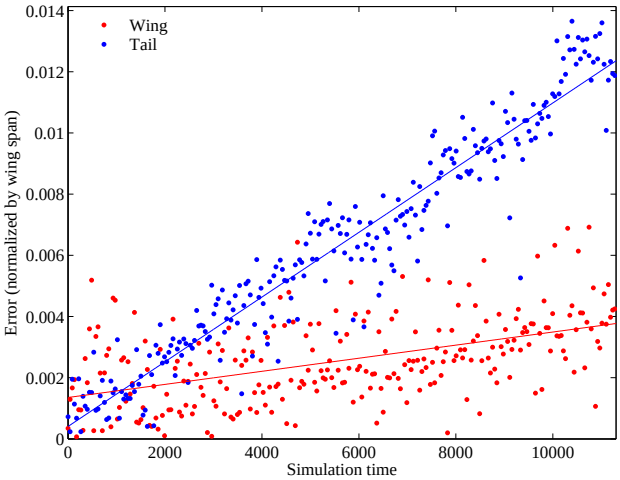


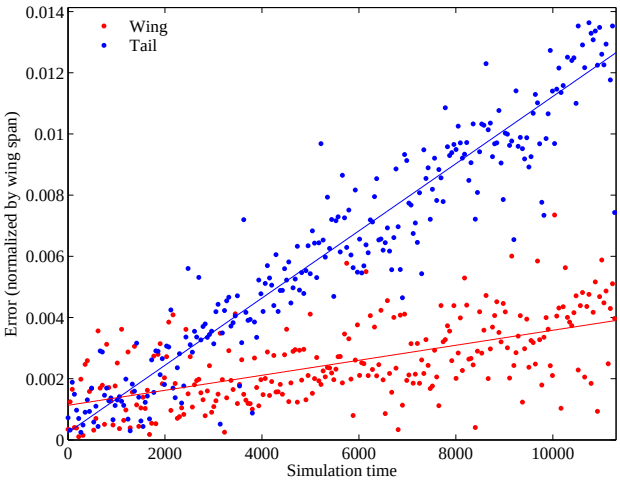
Figure 5.33: Lagrangian results for Large domain trajectory and trajectory error (compared to unbounded case)



(a) d03 h75

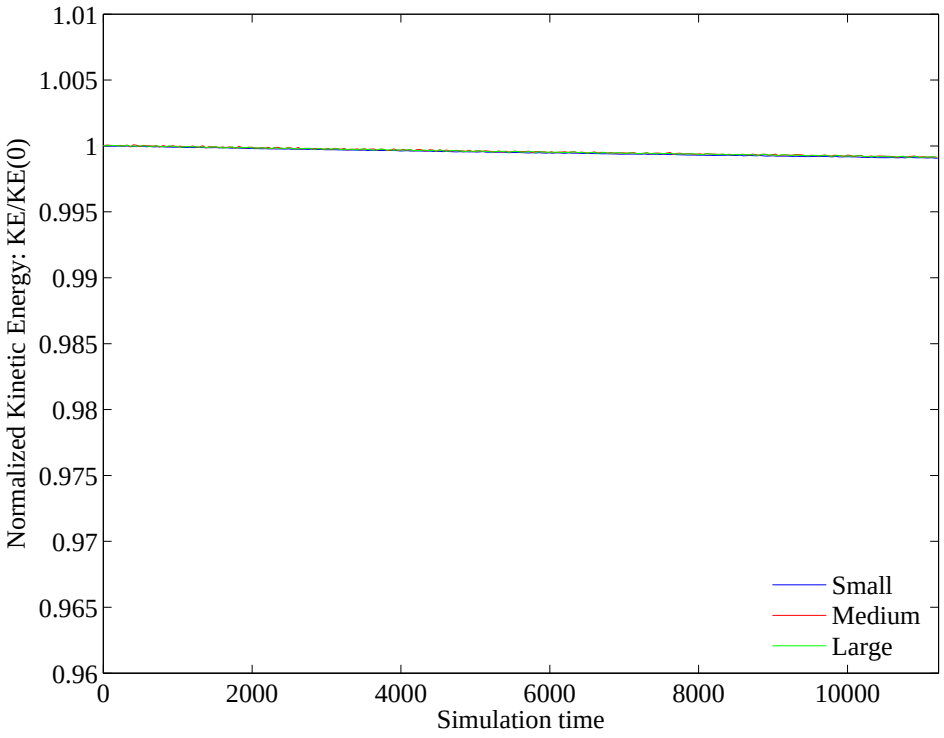


(b) d03 h150

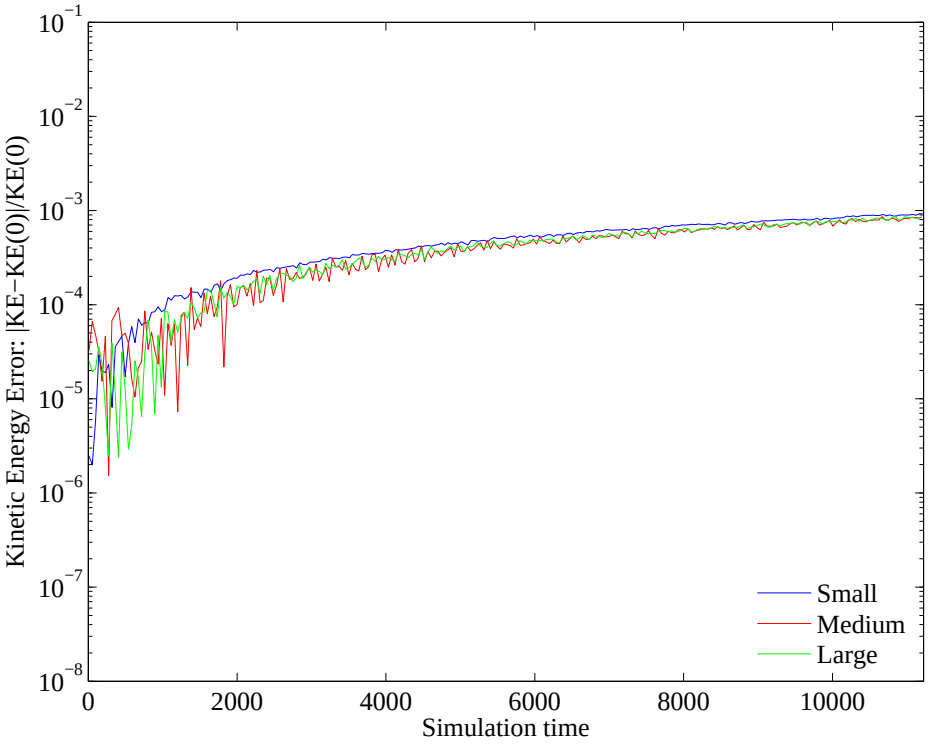


(c) d03 h300

Figure 5.34: Trajectory error: Small vs. Medium vs. Large domain (Degree 3, Coarse grid)

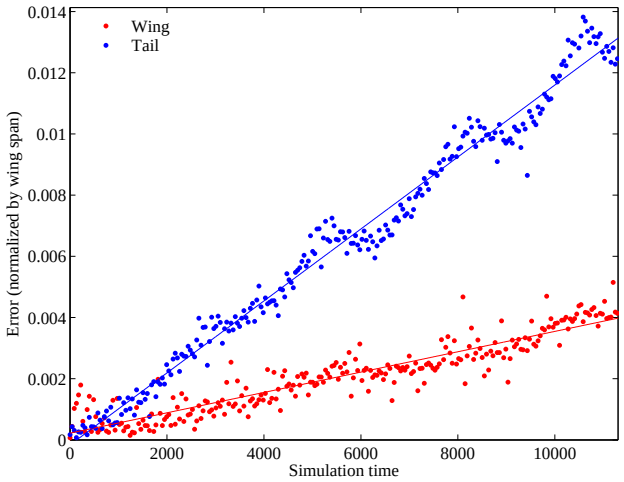


(a) Normalized Kinetic Energy

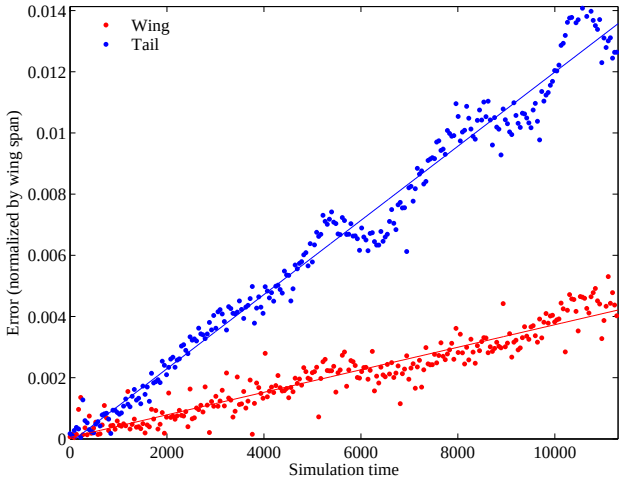


(b) Kinetic Energy Error

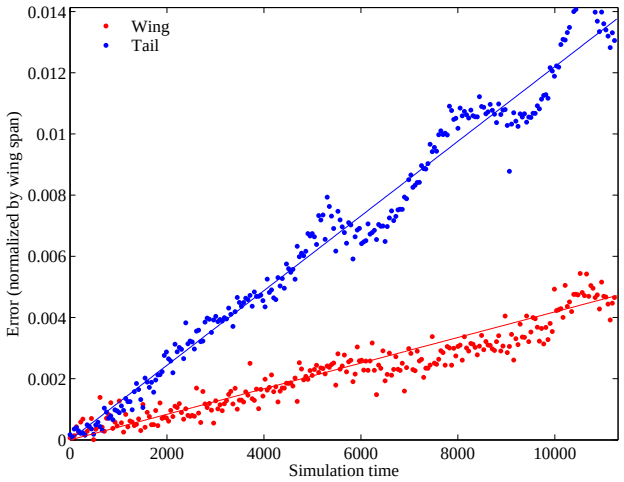
Figure 5.35: KE error: Small vs. Medium vs. Large domain (Degree 3, Coarse grid)



(a) d03 h150

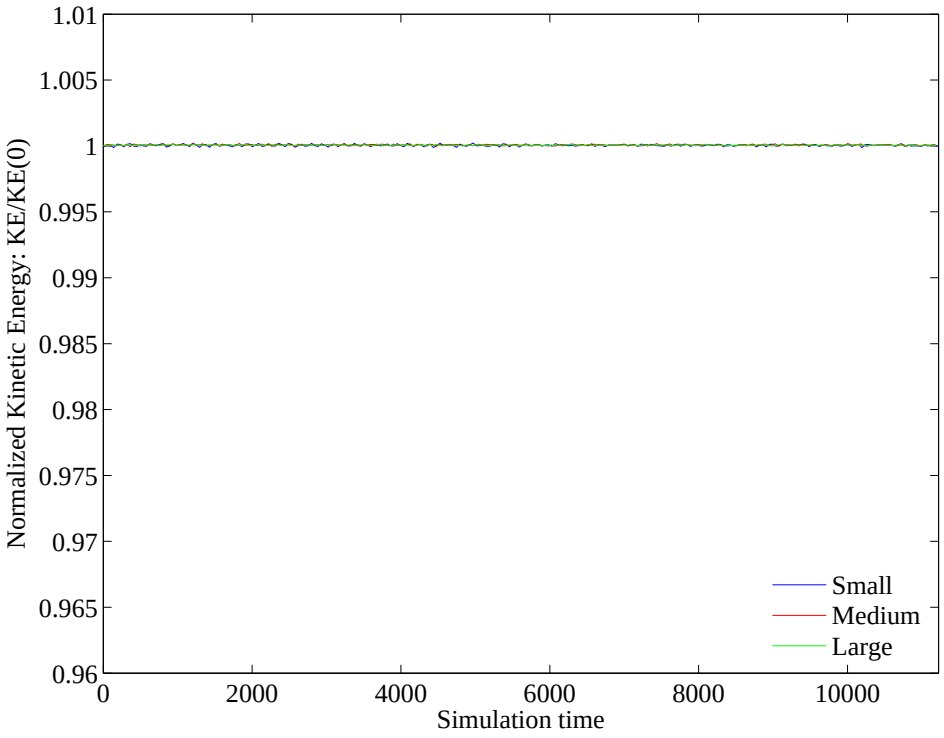


(b) d03 h300

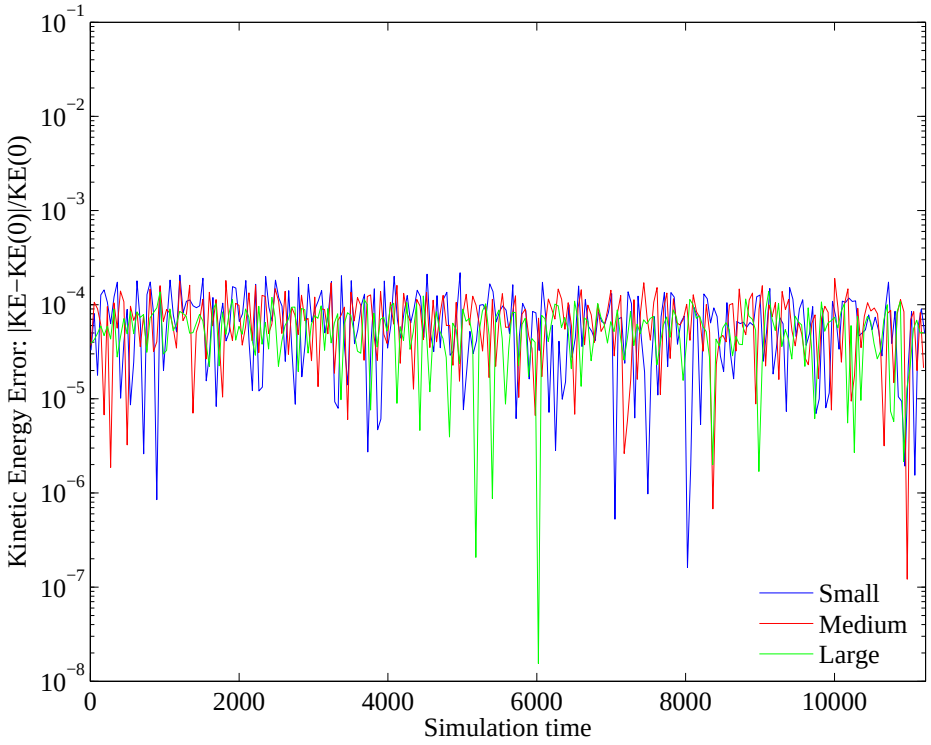


(c) d03 h600

Figure 5.36: Trajectory error: Small vs. Medium vs. Large domain (Degree 3, Fine grid)



(a) Normalized Kinetic Energy



(b) Kinetic Energy Error

Figure 5.37: KE error: Small vs. Medium vs. Large domain (Degree 3, Fine grid)

5.4.2 Coarsest Cell Density

As the other parameter with only two values for comparison, cell density is another good option for further exploration. As a review, the plots in Section 5.3 showed slightly higher trajectory error and slightly lower error kinetic energy for the fine cases when compared to the coarse density. Thus, rather than investigating an even finer density that may yield even higher trajectory error at a much higher cost, a density denoted as 'coarsest' makes for a more interesting third option that gives insight into the fine balance between cost and error. Plus, this draws on the strength of the DG method, which allows the use of larger cells with higher degree polynomials to provide the degrees of freedom instead of relying on high cell count with low polynomial degree.

Following in the pattern outlined in Chapter 4, the cell counts are found by halving those of the 'coarse' case. Instead of repeating all of the domain sizes and degrees with this coarsest grid, two were chosen selectively: degree 9 for the medium domain size and degree 3 for the large domain size. The reasoning behind this is that the small domain has been clearly found to be too small and constricting to the vortices, so no additional cases on that domain size would be helpful. Secondly, degree 3 on the large domain would complete the trio of coarsest, coarse, and fine densities for that size, and rather than also using degree 3 to complete the medium domain, degree 9 would add another complete and distinct set for the medium domain that could reveal different behaviors. The following complete sets for the two domain sizes are obtained:

Table 5.1: Expanded test cases

	Degree	Coarsest	Coarse	Fine
Medium Domain	9	30 x 48	60 x 96	120 x 192
Large Domain	3	150 x 240	300 x 480	600 x 960
DOF/unit		3.75	7.50	15.00

The results for the medium case (degree 9) are presented first in Figure 5.38, where it can be seen that despite having the fewest cells, the coarsest case exhibits the lowest error,

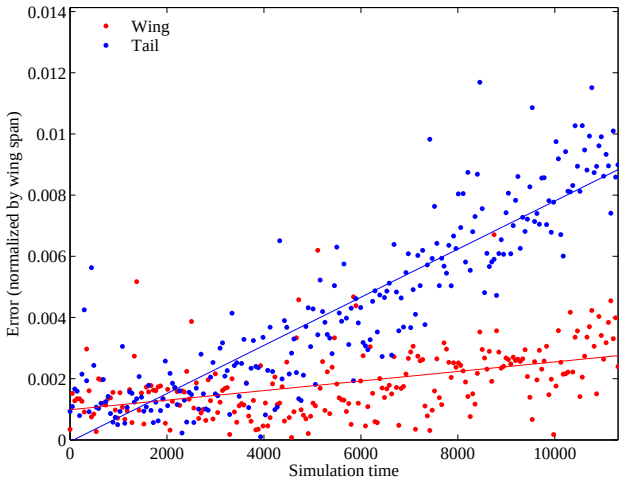
although the data is more sporadic and doesn't exhibit the same oscillatory pattern as the coarse and fine grids. However, considering that the 'coarsest' density correlates to a large cell size (when compared to Finite Volume) of over 2.5 units and still maintains low error is a huge benefit.

Again, the kinetic energy plots show that the large cells come with the tradeoff of more oscillatory normalized KE values and overall higher error compared to the the coarse or fine cell densities. It is certainly a tradeoff to accept higher KE error for lower trajectory error, but perhaps most importantly, finding that the degree 9 polynomial can accurately accommodate cell densities as low as 3.75 DOF/unit equates to a computational time of roughly one eighth that of the 'coarse' case. The decrease is from a four times reduction due to fewer cells and a further two times reduction because of the larger time step afforded by the larger cell size (which also allows half the overall steps to achieve the same final time).

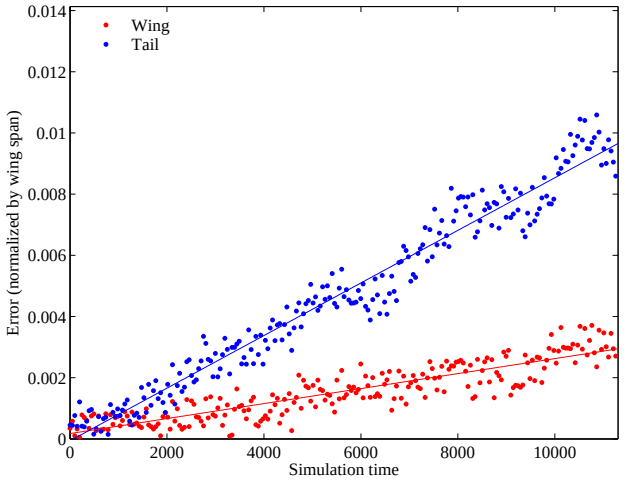
As stated in Chapter 1, one of the main goals in this work is to better understand how the various choices of parameters for the DG method interact and affect the quality of the solution. Up to this point, all test cases have provided varying solutions, but all were still acceptable. Figure 5.40, however, illustrates what happens when the limits of the DG method are exceeded. Although the 'coarsest' cell density worked (and even excelled) for the degree 9 case, it is clear that it is too coarse for a polynomial as low as degree 3. The error has no clear pattern at all and the normalized kinetic energy in Figure 5.41 (a) plummets and does not appear to be slowing down. The amount of energy in the system is not being held constant and is rapidly decaying. Figure 5.41 (b) shows more than an order of magnitude higher error when compared to the coarse case and over two orders of magnitude when compared to the fine case.

Figure 5.42 gives a closer look at the vortex trajectory. Small but noticeable deviations from the smooth path are present, while other cases show no visible trajectory difference. Ultimately, even though the simulation runs, 3.75 DOF/unit proves to be too coarse for a degree 3 polynomial and it is probably best to stay within the 'fine' cell density range of 15

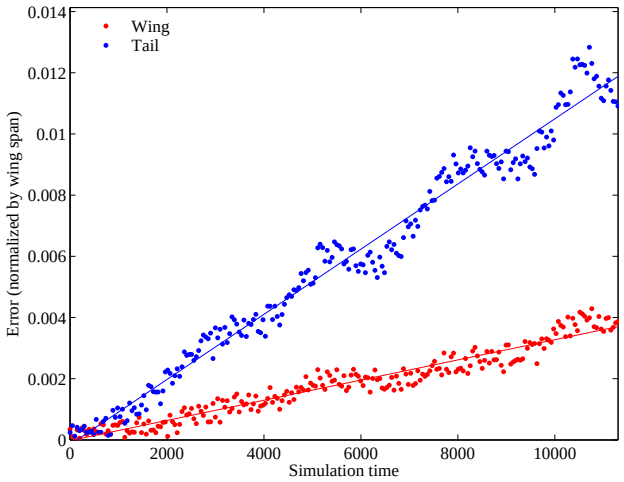
DOF/unit for lower polynomials like degree 3 to prevent KE deterioration. The results are bad enough that an argument can't be made that the lower cost justifies the higher kinetic energy error like with the degree 9 case.



(a) d09 h30

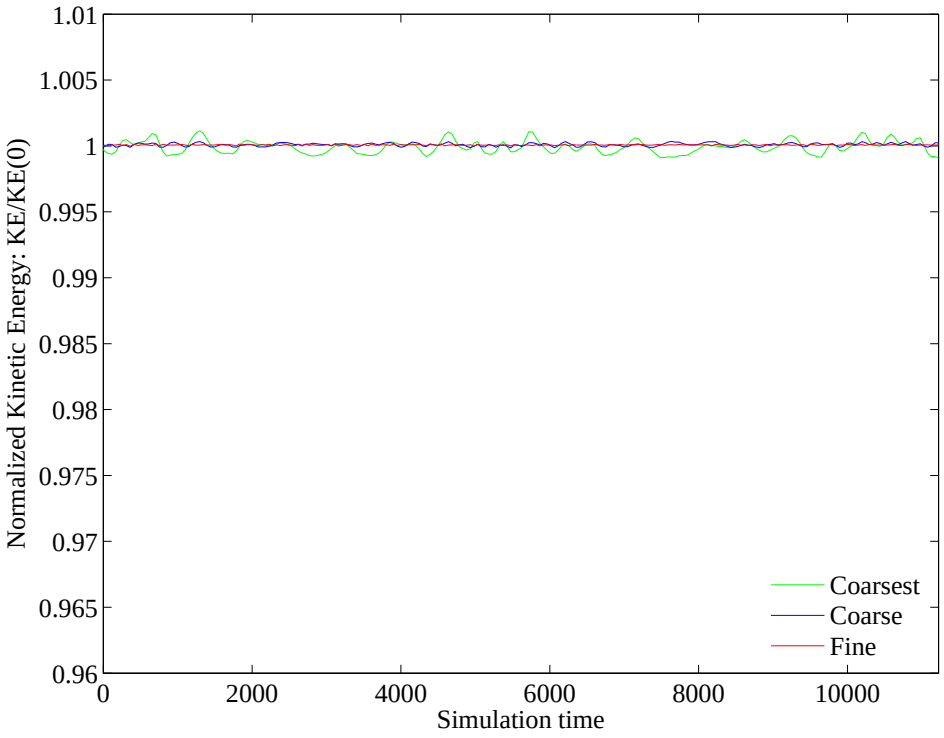


(b) d09 h60

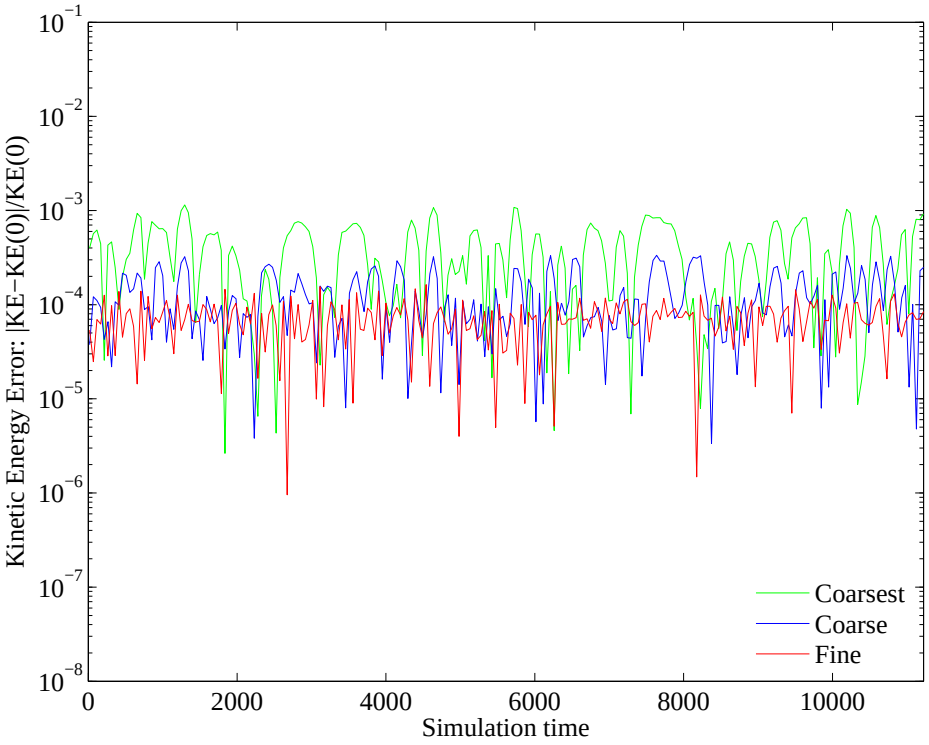


(c) d09 h120

Figure 5.38: Trajectory error: Coarsest vs. Coarse vs. Fine density (Medium domain, Degree 9)

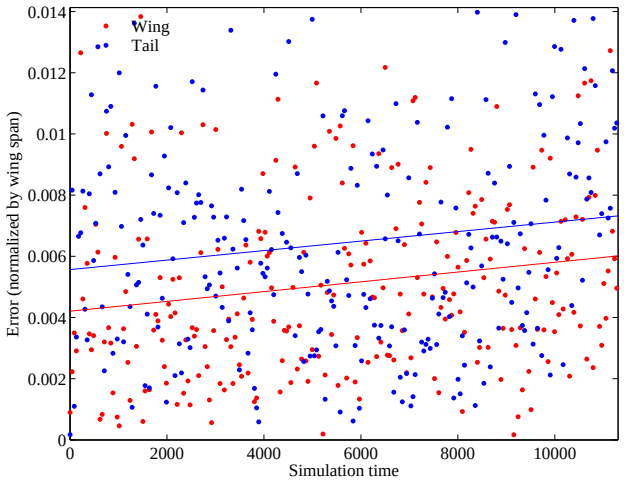


(a) Normalized Kinetic Energy

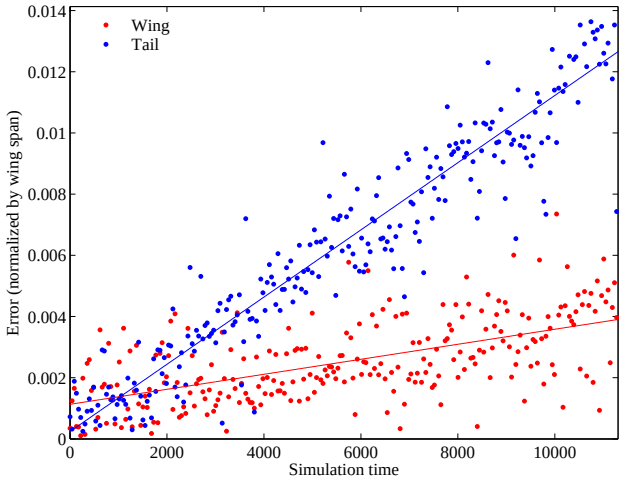


(b) Kinetic Energy Error

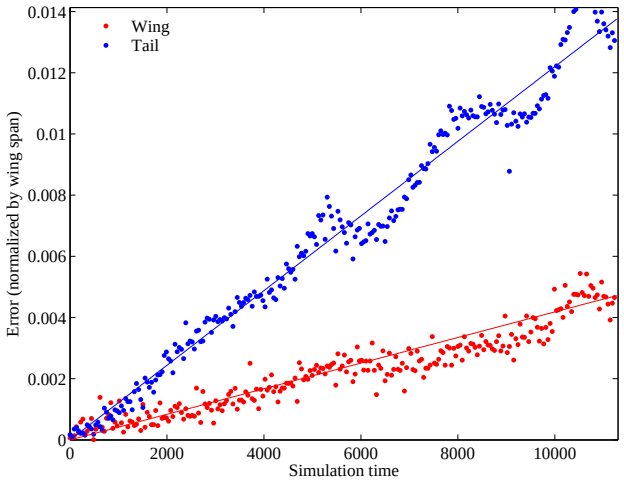
Figure 5.39: KE error: Coarsest vs. Coarse vs. Fine density (Medium domain, Degree 9)



(a) d03 h150

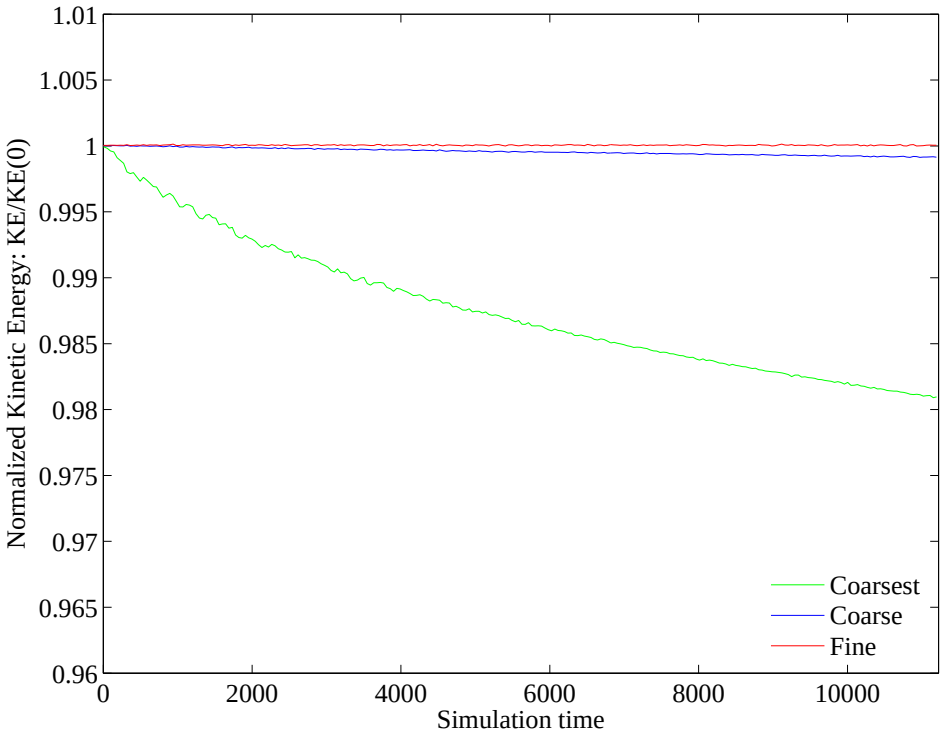


(b) d03 h300

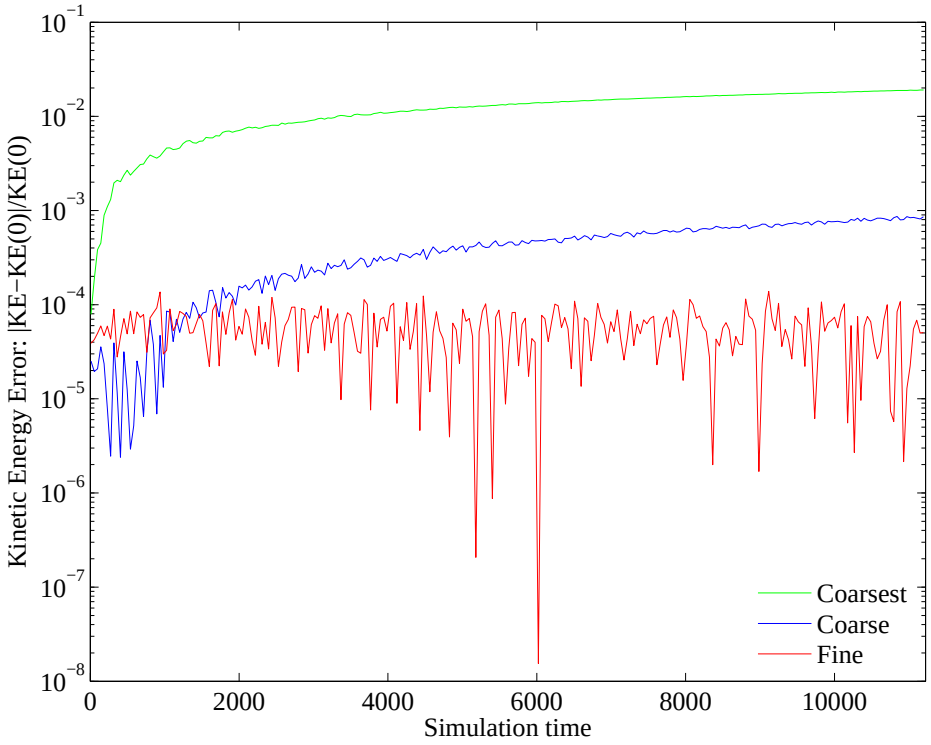


(c) d03 h600

Figure 5.40: Trajectory error: Coarsest vs. Coarse vs. Fine density (Large domain, Degree 3)



(a) Normalized Kinetic Energy



(b) Kinetic Energy Error

Figure 5.41: KE error: Coarsest vs. Coarse vs. Fine density (Large domain, Degree 3)

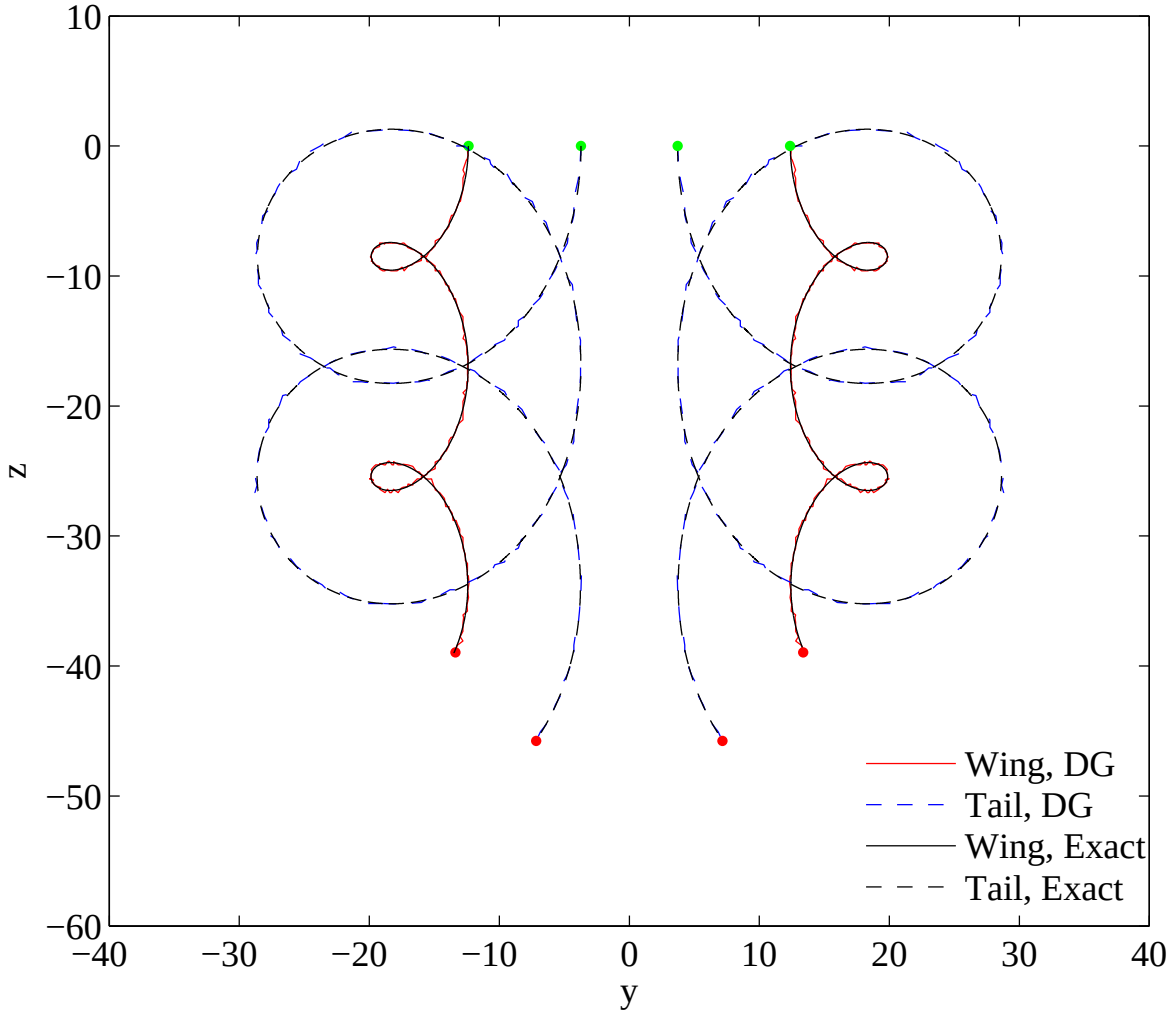


Figure 5.42: Trajectory for Coarsest density case (Degree 3, Large domain) compared to Lagrangian trajectory

5.5 Computational Time

Since one of the most important factors in any numerical method is its efficiency and how long it takes to compute a solution, Table 5.2 lists the wallclock times to run each case to 1000 steps. The cases were compiled identically using the Intel Fortran compiler and run on 8 processors. While it is clear that higher degree cases are more costly than lower degree, these numbers don't take into account that the cases with smaller cell sizes will also mandate a smaller time step - so 1000 steps for a coarse case will be twice as far into the simulation as the fine case.

Table 5.2: Run time for 1000 steps (seconds)

	Small			Medium			Large	
Degree	Coarse	Fine	Coarsest	Coarse	Fine	Coarsest	Coarse	Fine
3	18.63	78.06		83.54	342.52	82.70	324.22	1225.81
9	29.00	121.17	26.24	111.95	428.34			
14	45.26	183.95		181.22	719.69			

A better reflection of how long a particular case will take is given in Table 5.3, which uses the time step, wallclock time per step, and total number of steps to calculate a total wallclock time for the case (again on 8 processors). Each time is calculated assuming a total simulation time of 11,303. Figures 5.43 and 5.44 show the relative cost of increasing degree or density, respectively. While they both result in a higher overall cost, it is interesting to see that the method becomes more efficient as more cells are used. In general, the times reveal that increasing the domain size is roughly four times as costly (there are four times the cells) and doubling the cell density is about eight times as costly, meaning that doing both (e.g. going from medium coarse to large fine) costs about 32 times as much. This does not take into account that the lifting and stiffness matrices are sparse and the higher degree cases could benefit from coding that takes advantage of the sparseness to speed up matrix multiplication. It was excluded to maintain as much similarity between cases as possible and thereby make head-to-head comparisons more meaningful.

Table 5.3: Run time for full case (hours)

	Small		Medium			Large		
Degree	Coarse	Fine	Coarsest	Coarse	Fine	Coarsest	Coarse	Fine
3	2.64	22.14		11.85	97.15	5.87	45.98	347.67
9	7.40	61.86	3.35	28.58	218.68			
14	18.63	151.59		74.57	593.07			

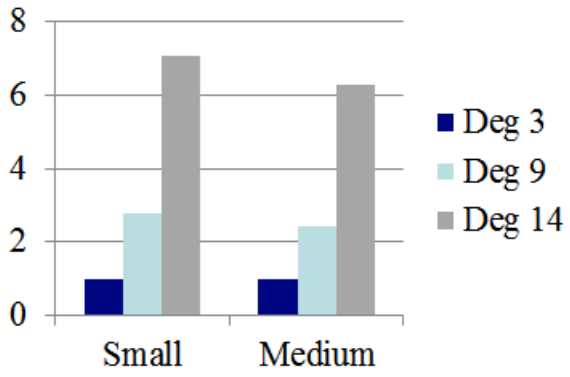


Figure 5.43: Effect of degree on computational time, normalized by time for degree 3 case

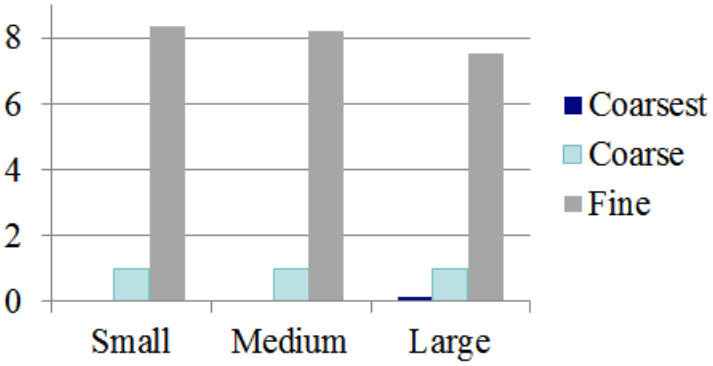


Figure 5.44: Effect of density on computational time, normalized by time for coarse density case

Chapter 6

Conclusions

Taken together, all of the results presented thus far have revealed some general trends for the DG method as they pertain to the interaction of two-dimensional vortices. They are as follows:

- Increasing the polynomial degree decreases overall error, but is more costly because it requires a smaller time step (and thus more total steps). A moderate polynomial degree of 9 seems to be a good trade-off between computational cost and accuracy.

- Domain size (boundary placement) is most influential on the solution itself and can cause huge differences in trajectory. Beyond that, increasing the domain size has little effect on error or kinetic energy when compared to the exact solution for that particular case. When compared to an unbounded case, however, a larger domain obviously produces better results.

- Lower cell density performs better, since the error is lower and is much cheaper, but comes with the tradeoff of slightly higher kinetic energy error. However, the lower limit for cell density depends on the degree of the polynomial, as 3.75 DOF/unit works well for degree 9 but produces poor results for degree 3.

- Doubling the domain size is proportionally not as computationally expensive as doubling the cell density, since doubling the cell density is roughly eight times more expensive whereas doubling the domain size is only about four times as expensive. The result is that decreasing the density, where possible, is the biggest time saver.

6.1 Future Work

While this work succeeded in its goal of outlining the effects each parameter has on the discontinuous Galerkin method for the specific case of counter-rotating vortices, further work is needed for confirmation and to continue towards the ultimate goal of better understanding complex fixed-wing and rotorcraft wakes. Some ideas for continuing research include:

- Perform similar studies with other simple flow fields to see if the same conclusions about the effects of polynomial degree, domain size, and cell density hold true. Some specific options include the same cases but with co-rotating and/or equal-strength vortices.
- Incorporate the selective artificial viscosity to dampen any spurious oscillations where needed.
- Implement other numerical flux functions to make sure results are not flux function dependent.
- Investigate even higher polynomial degrees and both higher and lower cell densities to determine if the trends continue linearly or if they behave in some other manner.
- Expand the capabilities to model three-dimensional flows and eventually more complex cases, while utilizing some of the lessons learned in this research (e.g. very fine grids will not only be costly, but also less accurate).

Bibliography

[1] Saffman, P. G., *Vortex Dynamics*, Cambridge University Press, 1992.

[2] Spalart, P. R., "Airplane Trailing Vortices," *Annual Review of Fluid Mechanics*, Vol. 30, No. 1, 1998, pp. 107–138.

[3] "FAR-Wake Project Description," `far-wake.irphe.univ-mrs.fr/IMG/project/project.html` [retrieved 7 November 2012].

[4] Dasey, T. J., Cole, R. E., Heinrichs, R., Matthews, M., and Perras, G., "Aircraft vortex spacing system (AVOSS) initial 1997 system deployment at Dallas/Ft. Worth (DFW) Airport," 1998.

[5] Hinton, D., "An aircraft vortex spacing system (AVOSS) for dynamical wake vortex spacing criteria," 1996.

[6] Proctor, F. H., *The NASA-Langley wake vortex modelling effort in support of an operational aircraft spacing system*, AIAA, 1998.

[7] Winckelmans, G., Cocle, R., Dufresne, L., and Capart, R., "Vortex methods and their application to trailing wake vortex simulations," *Comptes Rendus Physique*, Vol. 6, No. 45, 2005, pp. 467 – 486.

[8] Chatelain, P., Curioni, A., Bergdorf, M., Rossinelli, D., Andreoni, W., and Koumoutsakos, P., "Billion vortex particle direct numerical simulations of aircraft wakes," *Computer Methods in Applied Mechanics and Engineering*, Vol. 197, No. 1316, 2008, pp. 1296 – 1304.

[9] Reed, W. H. and Hill, T., "Triangular mesh methods for the neutron transport equation," *Los Alamos Scientific Laboratory Report, LA-UR-73-479*, 1973.

[10] Lesaint, P. and Raviart, P.-A., *On a finite element method for solving the neutron transport equation*, Univ. Paris VI, Labo. Analyse Numérique, 1974.

[11] Johnson, C. and Pitkaranta, J., "An analysis of the discontinuous Galerkin method for a scalar hyperbolic equation," *Mathematics of computation*, Vol. 46, No. 173, 1986, pp. 1–26.

[12] Hesthaven, J. S., "2008 Rocky Mountain Mathematics Consortium Summer School Lectures: Discontinuous Galerkin Methods," 2008, `www.cfm.brown.edu/people/jansh/page16/page36/page36.html` [retrieved 21 March 2012].

[13] Cockburn, B., Karniadakis, G. E., and Shu, C.-W., *The development of discontinuous Galerkin methods*, Springer, 2000.

[14] Hesthaven, J. S. and Warburton, T., *Nodal Discontinuous Galerkin Methods*, Vol. 54 of *Texts in Applied Mathematics*, Springer, 2008.

[15] Cockburn, B., "Discontinuous Galerkin Methods," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 83, No. 11, 2003, pp. 731–754.

[16] Cockburn, B. and Shu, C.-W., "Runge–Kutta discontinuous Galerkin methods for convection-dominated problems," *Journal of scientific computing*, Vol. 16, No. 3, 2001, pp. 173–261.

[17] Bassi, F. and Rebay, S., "A High-Order Accurate Discontinuous Finite Element Method for the Numerical Solution of the Compressible NavierStokes Equations," *Journal of Computational Physics*, Vol. 131, No. 2, 1997, pp. 267 – 279.

[18] Liou, M.-S., "A sequel to AUSM, Part II: AUSM$^+$-up for all speeds," *Journal of Computational Physics*, Vol. 214, No. 1, 2006, pp. 137–170.

[19] Liou, M.-S., "A Sequel to AUSM: AUSM$^+$," *Journal of computational Physics*, Vol. 129, No. 2, 1996, pp. 364–382.

[20] Lonfils, T., Cocle, R., Daeninck, G., Cottin, C., and Winckelmans, G., "Numerical investigations of end-effects associated with accelerated/decelerated wings: time-developing and space-developing simulations," *FAR-Wake Technical Report 1.1. 2*, Vol. 6, 2007.

[21] Desenfans, O., Lonfils, T., Daeninck, G., Cocle, R., Bricteux, L., Dufresne, L., and Winckelmans, G., "Numerical simulations of counter-rotating four-vortex systems," *FAR-Wake Technical Report 1.2. 2*, Vol. 2, 2007.

Appendix A

Lagrangian System Simulation

```
clear all

% Set values for vortex circulation and location

c = [ -1.0,  0.3, -0.3,  1.0 ] * 700./( 2.6*295.39 );
x = [ -1.0, -0.3,  0.3,  1.0 ] * 0.5*64.4/2.6;
y = [  0.0,  0.0,  0.0,  0.0 ] * 0.5*64.4/2.6;

xx = [-40,40];     % dimensions of plot box
yy = [-60,10];

dt = 1.;
Nstep = 11303;     % final simulation time

Nstage = 3;
rk = zeros(3,Nstage);

rk(1,:) = [ 1., 3./4., 1./3. ];
rk(2,:) = [ 0., 1./4., 2./3. ];
rk(3,:) = [ 1., 1./4., 2./3. ];

xsave = zeros( Nstep+1, length(c) );
ysave = zeros( Nstep+1, length(c) );
time = zeros( Nstep+1, 1 );

xsave(1,:) = x;
ysave(1,:) = y;
time(1) = 0.;

x0 = x;
y0 = y;
for step = 1:Nstep

        for stage = 1:Nstage
        [ u, v ] = induced_velocity( c, x, y );
        x = rk(1,stage)*x0 + rk(2,stage)*x + rk(3,stage)*dt*u;
```

```
        y = rk(1,stage)*y0 + rk(2,stage)*y + rk(3,stage)*dt*v;
        end

        time(step+1) = step*dt;
        x0 = x;
        y0 = y;
        xsave(step+1,:) = x;
        ysave(step+1,:) = y;
end
```

```
function [ u_f, v_f ] = induced_velocity( c, x, y )

n_rings=42;                 % the number of vortex rings to be used
rows=(2./3.)*n_rings;       % for Van-Wijngaarden accleration
cols=(1./3.)*n_rings;

N_vortex = length( c );
u = zeros(n_rings+1,N_vortex);
v = zeros(n_rings+1,N_vortex);

du=zeros(n_rings+1,1);
dv=zeros(n_rings+1,1);

u_f=zeros(1,N_vortex);
v_f=zeros(1,N_vortex);

g=1.781;
b=1.256;
rc=1;

xg=0;       % center of computation box
yg=0.;

xmax=40;    % small box
ymax=32;

Lx=2*xmax;
Ly=2*ymax;

du_vw=zeros(rows+1,n_rings+1,N_vortex);
dv_vw=zeros(rows+1,n_rings+1,N_vortex);


for m=1:N_vortex
```

```
for n=1:N_vortex

    for j=-n_rings:n_rings
        for i=-n_rings:n_rings
            if (j==0) && (i==0)  % means inside domain
                if( n~=m )       % ensures 2 diff vortices
                k=max(abs(i),abs(j))+1;    % vortex ring

                si=(-1)^abs(i);
                sj=(-1)^abs(j);
                x1=x(m);
                x2=(i*Lx+xg)-(xg-x(n))*si;
                dx=x1-x2;
                y1=y(m);
                y2=(j*Ly+yg)-(yg-y(n))*sj;
                dy=y1-y2;
                r2 = dx^2 + dy^2;
                f = c(n)*(1-exp((-b*r2)/(rc^2)))/( 2*pi*r2);

                u(k,m)=u(k,m)-si*sj*dy*f;
                v(k,m)=v(k,m)+si*sj*dx*f;

                end

            else                          % when outside domain
                k=max(abs(i),abs(j))+1;
                si=(-1)^abs(i);
                sj=(-1)^abs(j);
                x1=x(m);
                x2=(i*Lx+xg)-(xg-x(n))*si;
                dx=x1-x2;
                y1=y(m);
                y2=(j*Ly+yg)-(yg-y(n))*sj;
                dy=y1-y2;
                r2 = dx^2 + dy^2;
                f = c(n)*(1-exp((-b*r2)/(rc^2)))/( 2*pi*r2);
                u(k,m)=u(k,m)-si*sj*dy*f;
                v(k,m)=v(k,m)+si*sj*dx*f;

            end

        end
    end

end
```

```
end

            % Van Wijngaarden acceleration:

            for m=1:N_vortex
                du_cum_sum(:,m)=cumsum(u(:,m));
                dv_cum_sum(:,m)=cumsum(v(:,m));


                du_vw(1,:,m)=du_cum_sum(:,m);
                dv_vw(1,:,m)=dv_cum_sum(:,m);

                for row=2:rows+1
                    for col=1:n_rings
                        du_vw(row,col,m)=(du_vw(row-1,col,m)...
                            +du_vw(row-1,col+1,m))/2;
                        dv_vw(row,col,m)=(dv_vw(row-1,col,m)...
                            +dv_vw(row-1,col+1,m))/2;
                    end
                end
            end

                % final velocities after acceleration
                    u_f=du_vw(rows+1,cols+1,:);
                    v_f=dv_vw(rows+1,cols+1,:);

                    u_f=reshape(u_f,1,N_vortex);
                    v_f=reshape(v_f,1,N_vortex);


 end
```