UNCERTAINTY QUANTIFICATION FOR UNSTEADY FLUID FLOW USING ADJOINT-BASED APPROACHES

A DISSERTATION
SUBMITTED TO THE INSTITUTE FOR COMPUTATIONAL AND
MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Qiqi Wang

December 2008

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Parviz Moin) Principal Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Gianluca Iaccarino) Co-Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Antony Jameson)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Amin Saberi)

Approved for the University Committee on Graduate Studies.

Abstract

Uncertainty quantification of numerical simulations has raised significant interest in recent years. One of the main challenges remains the efficiency in propagating uncertainties from the sources to the quantities of interest, especially when there are many sources of uncertainties. The traditional Monte Carlo methods converge slowly and are undesirable when the required accuracy is high. Most modern uncertainty propagation methods such as polynomial chaos and collocation methods, although extremely efficient, suffer from the so called "curse of dimensionality". The computational resources required for these methods grow exponentially as the number of uncertainty sources increases.

The aim of this work is to address the challenge of efficiently propagating uncertainties in numerical simulations with many sources of uncertainties. Because of the large amount of information that can be obtained from adjoint solutions, we focus on using adjoint equations to propagate uncertainties more efficiently.

Unsteady fluid flow simulations are the main application of this work, although the uncertainty propagation methods we discuss are applicable to other numerical simulations. We first discuss how to solve the adjoint equations for time-dependent fluid flow equations. We specifically address the challenge associated with the backward time advance of the adjoint equation, requiring the solution of the primal equation in backward order. Two methods are proposed to address this challenge. The first method solves the adjoint equation forward in time, completely eliminating the need for storing the solution of the primal equation. The other method is a checkpointing algorithm specifically designed for dynamic time-stepping. The adjoint equation is still solved backward in time, but the present scheme retrieves the primal solution in reverse order. This checkpointing method is applied to an incompressible Navier-Stokes adjoint solver on unstructured mesh.

With the adjoint equation solved, we obtain a linear approximation of the quantities of interest as functions of the random variables describing the uncertainty sources in a

probabilistic setting. We use this linear approximation to accelerate the convergence of the Monte Carlo method in calculating tail probabilities for estimating margins and risk. In addition, we developed a multi-variate interpolation scheme that uses multiple adjoint solutions to construct an interpolant of the quantities of interest as functions of the uncertainty sources. This interpolation scheme converge exponentially to the true function, thus providing very accurate and efficient means of propagating of uncertainties and remains accurate independently of the locations of the available data.

Acknowledgements

I want to first thank my advisor, Professor Parviz Moin. In every discussion, his broad knowledge and deep insight has helped develop the directions for my research.

I also thank my co-advisor, Professor Gianluca Iaccarino. Throughout my Ph.D. study, he has provided helpful guidance, good company and many great ideas.

I gratefully acknowledge Doctor Frank Ham for his efforts in developing, maintaining and supporting the flow solver CDP. He has also provided many helpful suggestions and ideas.

I would like to thank Professor Antony Jameson for helpful advice. I also want to acknowledge Professor Amin Saberi, whose ideas have contributed to my work. Discussions with Professor George Papanicolaou and Professor Nasrollah Etemadi were also helpful.

I am deeply grateful to all those people who taught me mathematics, especially Professor Jihuai Shi at USTC, who taught me calculus, and Professor Shangzhi Li, who taught me linear algebra. They introduced me to the wonderful field of mathematics and cultivated my interest in it. I am also grateful to Professor Gene Golub, who taught me numerical analysis, Professor Peter Glynn, who taught me stochastic theories, and Professor Amir Dembo, who taught me probability theory.

I am indebted to my many friends and colleagues in Durand and in Building 500 for providing such an excellent environment for learning and research. I am especially grateful to Paul Constantine, David Gleich, Ali Mani, Yaser Khalighi, Tonkid Chantrasmi, Lee Shunn, Seongwon Kang and Mohammad Shoeybi for fruitful discussions.

Last, and most important, I wish to thank my family, especially my wife Rui Hu. This thesis would not be possible without her support. To her I dedicate my thesis.

Financial support for this work is provided by the Department of Energy under the Advanced Simulation and Computing (ASC) and the Predictive Science Academic Alliance Program (PSAAP).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Uncertainty is the only certainty there is, and knowing how to live with insecurity is the only security.

— John Allen Paulos

1.1 Uncertainty quantification: defining the next generation of computational simulations

Modern scientific research and engineering increasingly rely on computer simulations. The inability to produce an estimate of the uncertainties in the calculation is an important drawback of current computer simulation technology (Roache, 1997). For example, current weather forecast produce only a prediction of what the weather might be in the next few days. Due to the various errors made in the prediction process, the actual weather is notoriously likely to differ from the predicted weather (Palmer, 2000). Another example is in computational fluid dynamics simulations for aircraft design. Current CFD software can produce a performance profile for a proposed design, but it does not calculate how much different the predicted performance can be from the actual performance profile (Green *et al.*, 2002). The only way to be certain is to test a physical prototype in wind tunnels and in flight tests.

In contrast, if we could rigorously quantify the uncertainties in computer simulations, the results can be used in risk analysis and decision making. In weather prediction, for example, we want to calculate all possible scenarios, and estimate their likelihood. Forecasting

with quantified uncertainty enables one to make more informed decisions when planning weather-dependent activities (Roulston *et al.*, 2006). Returning to our second example above, we want to calculate possible differences between the actual and predicted performances. The designer can make more informed decisions based on the resulting uncertainty of the simulation results: he may choose to reduce the uncertainty by refining the computational model. If the level of uncertainty is acceptable, the design may be improved based on insights gained from the simulation. If both the level of uncertainty and the predicted design performance are satisfactory, the designer could finalize the design and begin working on a prototype. With the ability to accurately quantify errors and uncertainties, computer simulations will become much more powerful tools in science and engineering.

1.2 Two steps in uncertainty quantification: source and propagation

To quantify the uncertainties in the result of a simulation, one must understand both the sources of these uncertainties, and how uncertainties propagate through the simulation. The uncertainties in a simulation can originate from many sources, including

- uncertainties in initial conditions;
- uncertainties in boundary conditions;
- uncertainties in geometric parameters that describe the physical objects involved in the simulation;
- uncertainties from mathematical models that describe physical processes;

In addition, numerical errors in solving the mathematical models can be also treated as uncertainty. These errors include

- uncertainties in discretization errors of continuous differential equations that describe a physical process;
- uncertainties from finite precision floating point computations;
- uncertainties from the residual of iterative solvers.

In weather forecasting, meteorologists obtain the initial condition by feeding the measurements from weather observation stations into a data assimilation algorithm (Courtier *et al.*, 2002). Both the measurement errors from weather stations and the data assimilation process introduce uncertainties into the initial conditions (Pantano, 2007). In addition, various physical components such as turbulence, cloud formation and precipitation are described using inexact mathematical models. These models introduce uncertainty into the simulation. In solving the partial differential equations that describe the evolution of global weather, current computing technology can only afford a grid spacing in the order of kilometers, introducing a substantial amount of discretization and subgrid-scale physics uncertainty into the simulation. Finally, iterative solvers are often used in every time step of the partial differential equation solver. The residual of the iterative solver causes additional uncertainty in the predictions (Jameson, 2003). In a simulation of flow past an object, uncertainties in boundary conditions can result from inaccurate geometrical representations. Additional parametric uncertainties can result from inaccurate descriptions of the physical properties of the fluid (Trucano, 1998). The main task of uncertainty quantification is defining and quantitatively description of these uncertainties.

Several theories address the definition of uncertainty. These theories include probability theory (Green *et al.*, 2002) (Helton & Oberkampf, 2004), fussy set theory (Zimmermann, 1996) and evidence theory (Bae *et al.*, 2004) (Mourelatos & Zhou, 2004). In this report, we work under the framework of probability theory, which provides a solid and comprehensive theoretical foundation and offers the most versatile statistical tools. In contrast to the traditional, deterministic simulations, we describe uncertainties as randomness, and model the sources of uncertainties as random variables, random processes and random fields. To quantify the sources of uncertainties, we must specify the joint probability density function of all these random variables, processes and fields. This step is usually very problem-dependent. The methods involved in this step include statistical analysis, experimental error analysis and often expert judgment (Ellison *et al.*, 2000). Although how to quantify model uncertainties and numerical uncertainties is still a topic of current research (Wojtkiewicz *et al.*, 2001) (Draper, 1995), successful examples exist of quantifying the uncertainty sources for very complex engineering systems. For example, Bose & Wright (2006) were very successful in quantifying the uncertainties in the Martian atmosphere entry of the NASA Phoenix spacecraft.

Once the sources of uncertainties are quantified, we need to calculate how these uncertainties propagate through the simulation to the quantities of interest. These, also known as objective functions, are the main quantities to be predicted. They are functions of all the random variables that describe the sources of uncertainty. In simulating flow past an airplane, for example, the objective functions are the critical performance parameters, such as lift and drag (Roache, 1997).

In analyzing the propagation of uncertainties, we calculate how the objective functions are affected when the random variables describing the sources of uncertainties take different values (Isukapalli *et al.*, 1998). In a complex computational simulation, some sources of uncertainties, although present, have only negligible influence; in contrast, other sources of uncertainties may have a major impact on the variation of the objective functions and significantly contribute to the prediction error of the simulation (Yu *et al.*, 2006). An extreme case is the well-known "butterfly effect" in chaotic nonlinear dynamical systems: as the famous mathematician and meteorologist Edward Norton Lorenz hypothesized, *the flap of a butterfly's wings in Brazil could set off a tornado in Texas.* This is a vivid account for the following phenomenon: In a chaotic dynamic system, even a tiny amount of uncertainty in the initial condition can cause large deviations in its evolution. In fact, as uncertainties propagate through such a system, their magnitude often grows exponentially, until it becomes large enough to make simulation-based predictions completely unreliable. In this case, if the random variables describing the sources of uncertainties vary even slightly, they can produce a large variance in the objective functions (Yu *et al.*, 2006). Therefore, it is critical to quantify how the objective functions depend on the sources of uncertainties.

The final product of the uncertainty quantification process is a quantitative description of the likelihood in the values of the quantities of interest. It can only be obtained by combining our knowledge of the sources of uncertainties and the behavior of the objective functions with respect to these sources. In the probability theoretic framework, this quantitative description is a joint probability density function of the objective functions. The support of this joint probability density function, i.e., the space where the function is positive, describes all possible scenarios predicted by the computational simulation; in addition, the value of the probability density function indicates how likely each scenario is. This joint probability density function enables decision making based on risk analysis, removing the important limitations of deterministic computational simulations. Due to these benefits, we believe that uncertainty quantification will be the defining characteristic of the next

generation of computational simulation tools.

1.3 Methods for propagating uncertainty

As discussed in the previous section, uncertainty quantification involves two steps: determination of the uncertainty sources, and analysis of their propagation through the simulation. Denoting the random variables describing the sources of uncertainties as ξ, and the objective functions as $\mathbf{J}$, a mathematical abstraction of this two-step process is:

1. Quantify the joint probability density function of the vector ξ, which includes all random variables, discretized random processes and random fields that are used to describe the sources of uncertainty.

2. Given the probability density function of ξ, calculate the probability density function of $\mathbf{J}$. The function $\mathbf{J}$ is also called the response function or response surface.

Our research has been focused on the second part of the uncertainty quantification process, the propagation of uncertainty from the sources of uncertainties ξ to the objective functions $\mathbf{J}$. There have been extensive studies in this aspect of uncertainty quantification in the past few years. The common approaches are Monte Carlo methods (Bose & Wright, 2006), the polynomial chaos method (Xiu & Karniadakis, 2003), collocation-based methods (Mathelin *et al.*, 2005), and sensitivity-based methods (Putko *et al.*, 2001).

1.3.1 Monte Carlo methods

In Monte Carlo methods, a sequence $\xi_1, \xi_2, \ldots, \xi_n$ is sampled according to the probability distribution of ξ, which is obtained from the first step of uncertainty quantification. Deterministic simulations are performed for each sampled ξ_i to obtain the objective function $\mathbf{J}(\xi_i)$. The empirical density function is obtained by plotting the histogram of the objective functions. As the number of samples increases, this empirical density function converges to the probability density function of the objective functions. In reality, we can perform simulations on a finite number of samples; the resulting empirical density function is used as an estimate of the probability density function (Kuczera & Parent, 1998).

There are several advantages of Monte Carlo methods. First, implementing Monte Carlo methods is generally simple once a deterministic solver is available. The implementation

only involves a sampling method for ξ, the execution of the deterministic solver for calculating $\mathbf{J}$, and the collection of statistics from the resulting samples of $\mathbf{J}$. Second, Monte Carlo methods are naturally insensitive to the dimensionality of the parameter space, and does not have the "curse of dimensionality". Third, because the deterministic solver can be run on different processors for different samples ξ_i, Monte Carlo methods are inherently parallel. By using a Monte Carlo method, we can take full advantage of a parallel computer even if the deterministic solver runs only on a single processor. In addition, Monte Carlo methods are generally very robust, due to their simplicity. They usually work whenever the underlying deterministic solver works.

On the other hand, the major drawback of Monte Carlo methods is their slow rate of convergence. Although there are many techniques to increase its rate of convergence, such as variance-reduction methods using control variates and importance sampling techniques, the ultimate rate of convergence of all Monte Carlo methods is governed by the central limit theorem, and is of the order of $n^{\frac{1}{2}}$ (where n is the number of samples) without exception. The implication of this slow convergence is high computational cost, and relatively large approximation error of the empirical density function of $\mathbf{J}$. Due to this limitation, Monte Carlo is only suitable when the desired accuracy is not very high; for example, when the result of the first step of uncertainty quantification, the probability distribution of the sources of uncertainties, is not very accurate. Since the inaccurate quantification of the uncertainty sources dominates the error in the quantified uncertainty, calculating the propagation of uncertainty with high precision would not be prudent. In this situation, Monte Carlo can be the method of choice. However, if the sources of uncertainties can be accurately quantified, and an accurate probability distribution of the objective functions is desirable, using Monte Carlo methods can be very costly in terms of computational resources (Rubinstein, 1981) (Hammersley & Handscomb, 1964).

1.3.2 Intrusive polynomial chaos

In contrast to slow-converging Monte Carlo methods, polynomial-chaos-based methods are theoretically proven to converge exponentially fast for smooth objective functions (Xiu & Karniadakis, 2003). This method chooses a set of orthogonal multi-variate polynomials $\{\Phi_i(\xi) \mid i \in \mathbb{N}^d\}$ as an infinite dimensional basis in the random space, with d as the number of components of ξ, and $i = (i_1, \ldots, i_d)$. With this basis, each random variable in the computational simulation is represented using an infinite series known as polynomial chaos

expansion. In a time-dependent partial differential equation, the solution can be represented as

$$u(x,t;\xi) = \sum_{i \in \mathbb{N}^d} u_i(x,t)\, \Phi_i(\xi),$$

where each u_i is deterministic. In computation, a truncated series is used

$$u(x,t;\xi) \approx \sum_{\substack{i \in \mathbb{N}^d \\ |i| \leq P}} u_i(x,t)\, \Phi_i(\xi), \tag{1.1}$$

where P is the order of the expansion. Using a Galerkin projection in the random space, a set of coupled partial differential equations governing the evolution of $u_i(x,t)$ can be derived from the partial differential equation governing the evolution of $u(x,t;\xi)$. These coupled partial differential equations are then solved to obtain each $u_i(x,t)$. Using similar Galerkin projections, the objective function can also be approximated using truncated polynomial chaos expansions

$$\mathbf{J}(\xi) \approx \sum_{\substack{i \in \mathbb{N}^d \\ |i| \leq P}} \mathbf{J}_i\, \Phi_i(\xi),$$

and the coefficients $\mathbf{J}_i$ can be calculated from $u_i(x,t)$. This expansion of the objective functions and its coefficients is the final product of the polynomial chaos analysis (Xiu & Karniadakis, 2002). Many important statistics of the objective functions, such as mean and variance, can be analytically calculated from this expansion. More complex statistics can be calculated by sampling the response surface defined by (1.1), a process that is very fast because (1.1) is easy to evaluate.

Under certain conditions, the error of the truncated series expansion is proven to decrease exponentially fast with respect to the polynomial order P. Experimentally, intrusive polynomial chaos method has been shown to be by far superior to Monte Carlo methods, especially on relatively simple problems. When the required precision is high, polynomial chaos can dramatically reduce the required computational effort (Ghanem & Spanos, 1990).

In more complex problems, however, polynomial chaos methods have two major limitations. The first is the infamous "curse of dimensionality". For a polynomial chaos expansion truncated at order P, the number of terms in the expansion is $\dfrac{(P+d)!}{P!\,d!}$. The cost of solving the polynomial chaos system grows at least proportionally to the number of terms in the truncated polynomial chaos expansion (Rubinstein & Choudhari, 2005). As a result, for

the polynomial order P, the cost of the polynomial chaos method grows very fast, often exponentially with respect to d, the dimensionality of the random space. In complex problems, this dimensionality may be hundreds or even thousands, making polynomial chaos method with $P > 2$ totally infeasible. This limitation makes its advantage, the exponential convergence with respect to P, irrelevant in these practical applications.

The second limitation is due to the intrusiveness of this method (Hosder *et al.*, 2006). In intrusive polynomial chaos methods, the original equation governing the evolution of $u(x, t; \xi)$ is transformed into a set of coupled equations that govern the evolution of the coefficients of the truncated polynomial chaos expansion $u_i(x, t)$. These new equations often have different characteristics than the original equation, and can be much more complicated. As a result, using the intrusive polynomial chaos method requires writing a new solver. For complex engineering problems, this involves a very large amount of coding work. In addition, if the new equations are not discretized properly, numerical instabilities may occur (Debusschere *et al.*, 2005). This consideration makes additional work in numerical analysis necessary before writing the new solver. These two barriers limit the application of intrusive polynomial chaos to complex problems with many degrees of uncertainty.

1.3.3 Collocation-based methods

Collocation-based methods are designed to remove the second limitation of intrusive polynomial chaos methods (Mathelin *et al.*, 2005). According to the probability distribution of random variables ξ describing the sources of uncertainty, collocation methods select a multi-dimensional grid $\xi_1, \ldots, \xi_Q$. Commonly used grid types include tensor product grids and Smolyak sparse grids constructed as combinations of 1-D grids, such as Gauss quadrature grid or Clenshaw–Curtis grid (Cheng & Sandu, 2007). A deterministic solver is used to calculate the objective function $\mathbf{J}(\xi_i)$ for each point on the multi-dimensional grid. Using quadrature methods on the multidimensional grid, many statistics of the objective function can be directly computed from its values on the grid. Alternatively, one can represent the objective function using a polynomial chaos expansion, whose coefficients can be computed using similar quadrature methods. This approach is commonly referred to as non-intrusive polynomial chaos method (Hosder *et al.*, 2006).

The main advantages of collocation methods are exponential convergence and non-intrusiveness (Babuska *et al.*, 2007). Similar to the polynomial chaos, the error of collocation-based methods decreases exponentially as the order in each dimension (if a tensor product

grid is used) or the total order (if a sparse grid is used) increases. In addition, like Monte Carlo methods, collocation methods solve a deterministic problem at each grid point. If a deterministic solver is available, collocation methods do not modify its source code, and do not require developing a new solver. Therefore, using the non-intrusive collocation methods saves significant effort in software engineering and coding.

Despite these benefits, collocation methods suffer from the same curse of dimensionality as intrusive polynomial chaos does. In fact, compared to intrusive polynomial chaos, the computational cost tends to grow even faster with respect to the dimensionality of the random space. The Smolyak sparse grid is designed to alleviate this problem, but the growth of cost with respect to dimensionality is still at least as fast as intrusive polynomial chaos (Cheng & Sandu, 2007). This curse of dimensionality limits both polynomial chaos and collocation-based methods to relatively simple problems in which only a small number of uncertainty sources are considered.

1.3.4 Sensitivity-based methods

Sensitivity-based methods are a class of uncertainty propagation methods that use the derivatives of the objective function with respect to the random variables describing the sources of uncertainty, i.e., the sensitivity derivative, $\frac{\partial \mathbf{J}}{\partial \xi}$. Adjoint-based methods, which are the primary focus of this work, is a subclass of sensitivity-based methods. The sensitivity derivative can be calculated using three methods: finite differences, tangent linear analysis, and adjoint analysis (Eldred *et al.*, 2002). In finite difference method, the objective function at two nearby points in the random space $\mathbf{J}(\xi_0)$ and $\mathbf{J}(\xi_0+\Delta\xi)$ are calculated using deterministic solvers. The difference between the values of the objective function is used to approximate its derivative. In order to get the full gradient, at least $d+1$ deterministic calculations must be performed. The finite difference method is easy to implement and non-intrusive, but may produce inaccurate derivatives.

In contrast, the tangent linear method derives a linear equation that governs the evolution of the sensitivity. By solving this tangent linear equation, the exact derivatives of the objective function can be calculated. Similar to the finite difference method, at least d instances of the tangent linear equation must be solved with different initial conditions in order to obtain the full gradient.

The adjoint method also computes the exact gradient. In this method, the adjoint linear equation is solved instead of the tangent linear equation. The number of adjoint equations

that must be solved to obtain the full gradient equals to the number of objective functions, but is independent of d, the dimensionality of the random space. In complex problems, the number of objective functions is often much smaller than the dimensionality of the random space. As a result, the adjoint method can be much more efficient than the other two. Accordingly, in this work, we focus on adjoint-based methods.

Sensitivity-based methods are best known for their efficiency in terms of computational resources (Eldred *et al.*, 2002). There are several approaches that use the gradient $\frac{\partial f}{\partial x}$ to analyze the propagation of uncertainty. Each method has its own benefits and limitations. The perturbation method approximates the objective function using a first- or second-order truncated Taylor series, which can be constructed from the derivatives. The probability distribution of the objective function is approximated by the probability distribution of its low-order approximation, which can be directly calculated from the probability distribution of the sources of uncertainties (Putko *et al.*, 2001). This methods is simple and efficient, but is limited to the cases when the magnitudes of uncertainties are sufficiently small. When large magnitude of uncertainty is present, the first-order truncated Taylor series is not a valid representation of the objective function. In this case, the linear perturbation method must be combined with other methods, such as collocation methods, in order to produce accurate results. Another method that uses the sensitivity derivative is commonly referred to as a two-step hybrid approach. In the first step of this method, the sensitivity derivative is calculated, and is used to determine which uncertain parameters have significant impact on the objective function. In the second step, the uncertain parameters that are considered insignificant are discarded, reducing the dimensionality of the random space. A Monte Carlo, polynomial chaos or collocation method is then used on the reduced random space (Bose & Wright, 2006). This approach is most suitable when the dimension of the random space is large, and the magnitude of uncertainty is large only for a small number of random parameters.

In addition to these two methods, we propose two new uncertainty propagation methods using the sensitivity gradient. The adjoint accelerated Monte Carlo is suitable for estimating the tail probabilities. The adjoint-based interpolation is an efficient and accurate method of capturing the nonlinearity in the propagation of uncertainties. By replacing the objective function with this interpolation approximation, this method can be combined with most uncertainty propagation methods to efficiently obtain accurate results.

1.4 Propagating uncertainty in unsteady fluid flow using the adjoint method

This report focuses on propagating uncertainty in unsteady fluid dynamics problems using sensitivity gradient calculated by solving the adjoint equation. We address two main problems:

1. How to solve the adjoint equation for unsteady fluid flow.
2. How to use the sensitivity gradient obtained from the adjoint solution for propagation of uncertainty.

1.4.1 Solving the adjoint equation for unsteady fluid flow

Unsteady fluid flows are described with time-dependent partial differential equations, the Navier-Stokes equations. For unsteady problems, the adjoint equation is a partial differential equation that has a similar structure as the primal equation; the major differences are:

1. The adjoint equation is a linear partial differential equation, while the primal equation (such as the Navier-Stokes equations) is usually nonlinear. This linearity can make the adjoint equation easier to solve than its primal equation.
2. When the primal equation is nonlinear, the coefficients in the adjoint equation depend on the solution of its primal solution. Thus, the solution of the primal equation is required before the adjoint equation can be solved.
3. The adjoint equation is marched backward in time. When the adjoint equation is derived for time-dependent problems, the calculations start from the terminal solution rather than an initial condition. In order to calculate the sensitivity derivatives, the adjoint solution must be computed at each time step. Therefore, in solving the adjoint equation, we usually start at the very last time step, at which the terminal condition is given.

The second and third characteristics combine to pose a unique challenge for efficient solution of unsteady adjoint equations. The adjoint solution process marches backward, and the solution at each time step requires the solution of the primal equation at the same

time step. Therefore, the solution of the primal equation is needed in a time-reverse order, which leads to a serious demand on computer memory (Griewank & Walther, 2000).

We present two solutions to this problem. The first solution is changing the time-reversed nature of the adjoint solution process. By using a Monte Carlo linear solver (Srinivasan, 2003) (Okten, 2005), the adjoint equation can be solved forward in time. Using this method, the adjoint equation at each step can be obtained immediately after solving the primal equation at the same step. The solution of the primal equation at previous time steps can then be discarded. We demonstrated this method for the scalar transport partial differential equation, though more issues remain to be resolved for systems of coupled partial differential equations such as the Navier-Stokes equations.

The second solution is using a dynamic checkpointing method (Charpentier, 2001). The basic idea of checkpointing methods is to solve the entire primal equation first, and store its solution at selected time steps called checkpoints. When the adjoint equation is integrated backward in time, the solutions at corresponding time steps are calculated by re-solving the primal equation starting from the nearest checkpoint (Griewank & Walther, 2000). The dynamic checkpoint scheme described in this work is for situations when the number of time steps is not known in advance. It minimizes the maximum number of recalculations for each time step, and guarantees an efficient calculation of the adjoint equation when memory storage is limited. In contrast to previous online checkpointing methods (Heuveline & Walther, 2006) (Stumm & Walther, 2008) (Hinze & Sternberg, 2005), our scheme has provable performance bounds and works for arbitrarily large number of time steps.

Based on our checkpointing scheme, we have developed an adjoint solver for the unsteady incompressible Navier-Stokes equations. The adjoint equation for unsteady Navier-Stokes equations has been derived and is solved in short time horizon for flow control purposes (Bewley *et al.*, 2001) (Bewley, 2001). Because the dynamic checkpointing technique is used, the adjoint solver we have developed is suitable for long time integration. We solve the unsteady adjoint Navier-Stokes equations on unstructured mesh, making it suitable for studying a large variety of fluid flows in complex geometries.

1.4.2 The advantages of using adjoint solution for propagating uncertainties

The main advantage of using the adjoint solution is that it is a very cost effective way of obtaining information about the objective function. The entire gradient $\nabla \mathbf{J}$ of each

objective function with respect to all components of ξ can be calculated by solving a single adjoint equation (Reuther *et al.*, 1999). When the dimension of ξ is large, i.e., when a large number of random variables are needed to describe the sources of uncertainty, the gradient of $\mathbf{J}$ reveals much more information about $\mathbf{J}$ and its statistics than the value of $\mathbf{J}$. For example, consider a single objective function, with the dimension of ξ equal to 1000, evaluating the value of $\mathbf{J}$ at a single point produces only one number, $\mathbf{J}(\xi)$ at that point. However, if the gradient of $\mathbf{J}$ can be calculated at the same point, we obtain 1000 numbers indicating the variability of $\mathbf{J}$ along each dimension of ξ. Moreover, if the gradient can be computed with less than 1000 times the computational cost of computing the function value, and the information from the gradient can be fully utilized in calculating the statistics of the objective function, then computing the gradient at each point would be a more efficient way to calculate statistical information of the objective function (Eldred *et al.*, 2002). Accordingly, in this work, we have developed two methods of using the adjoint sensitivity gradient in propagating uncertainties.

The first method is the accelerated Monte Carlo method presented in Chapter 5. It is designed to calculate tail probabilities for risk analysis. By using control variate and importance sampling techniques based on the adjoint sensitivity gradient, we can concentrate the samples around the tail of the probability. This reduces the variance of Monte Carlo methods and accelerates their convergence.

The second method is an adjoint-based multi-variate interpolation method. In this method, we approximate the objective function $\mathbf{J}(\xi)$ with an interpolant $\tilde{f}(\xi)$, which is constructed from the value and gradient of the objective function at a finite number of points. The interpolation grid can be arbitrary, and the error of the interpolation approximation decreases exponentially as more points are used. The statistics and density function of the objective functions are then approximated as the statistics and density function of the surrogate objective functions $\tilde{f}$, which can be calculated efficiently using a Monte Carlo method. Because the adjoint gradient is used, this method is efficient; because the choice of the interpolation grid is arbitrary, this method is also flexible. This method is also accurate due to the spectral convergence of the interpolation scheme.

1.5 Accomplishments

- Developed a Monte Carlo adjoint solver, the first unsteady adjoint solver that runs forward in time.
- Developed the dynamic checkpointing scheme, the first optimal checkpointing scheme that works for an arbitrary number of time steps, and with proven bounds on its performance.
- Developed an adjoint solver for CDP, the first long time integration adjoint solver for hybrid unstructured mesh, unsteady Navier-Stokes solver.
- Developed a method that can dramatically accelerate Monte Carlo convergence for risk analysis.
- Invented an interpolation scheme that converges exponentially fast and works on arbitrary grids. It is also the first multi-variate interpolation scheme for arbitrary scattered data and matches the gradient at interpolation nodes.

Chapter 2

Monte Carlo Adjoint Solver

2.1 Introduction

Although adjoint-based methods have a long history in computational sciences and engineering (Bryson *et al.*, 1969), computing the solution of an unsteady adjoint equation is difficult: when the original problem is a time-dependent (unsteady) problem, solving its adjoint equation is a backward-time procedure, and requires the full trajectory of the primal solution to be stored in memory. This trajectory is formed by the solution of the original problem at all time steps, and is often too large to store in memory. Griewank (1992) proposed a very interesting iterative checkpointing scheme called *Revolve* for dealing with this problem. In his scheme, storing the full trajectory is avoided by iteratively solving the original problem. This idea made solving adjoint equations possible for larger unsteady problems. Since then, a number of similar schemes have been proposed (Charpentier, 2001) (Walther & Griewank, 2004). Nevertheless, all of these schemes are significantly more expensive than solving the original problem in terms of both memory requirements and computational time. As a result, if the original problem is very large in terms of the number of degrees of freedom, solving the adjoint equation is still prohibitively expensive.

In this chapter, we propose a new method for solving the adjoint equation for unsteady problems. Instead of computing an exact solution, we use a Monte Carlo method to approximate the solution. This method samples Markovian random walks in the space-time structure of the original problem and estimates quantities of interest from these samples. The method builds upon Monte Carlo linear solvers for general linear systems (Forsythe & Liebler, 1950) (Dimov, 1998) (Tan, 2002) (Okten, 2005) and some related works (Pharr

& Hanrahan, 2000). We found that the Monte Carlo method is particularly suitable for solving unsteady adjoint equations. *In contrast to traditional methods used for solving the adjoint equation, this method is a forward-time procedure.* Neither storing the trajectory nor iteratively solving the original problem is required. Therefore, the memory requirement and computational time of our Monte Carlo method are a constant multiples of the original problem.

In the remainder of this chapter, Section 2.2 introduces the unsteady adjoint equation. Section 2.3 describes the traditional exact solution methods. In Section 2.4, we introduce our Monte Carlo method for solving adjoint equations. Because many large problems arise from discretized partial differential equations, we describe our method specifically for this case. The error and variance of this method is analyzed in Section 2.5. In Section 2.6, we apply our Monte Carlo adjoint solver to the Burgers' equation. In Section 2.7, we show the results of several numerical experiments with Burgers' equation.

2.2 The unsteady adjoint equation

Consider a state vector u controlled by a control vector η via constraint or governing equation $\mathcal{R}$. The objective function $\mathcal{F}$ is defined on the space of u and η. We denote this by

$$\begin{aligned} &\mathcal{F}(u,\eta) \\ &\mathcal{R}(u,\eta) = 0, \end{aligned} \tag{2.1}$$

where $\mathcal{F}$ is a scalar function, and $\mathcal{R}$ is a vector function with $dim(\mathcal{R}) = dim(u)$. In the context of an adjoint equation, system (2.1) is often referred as the *original problem*.

Many engineering applications fit this problem type. Generally, $\mathcal{R}$ is an algebraic or differential equation modeling a physical system, u is a vector describing the state of the system, and η is a vector composed of a set of control parameters. In the context of computational fluid dynamics, most problems concern objects in a flowfield. In these problems, the constraint $\mathcal{R}$, the Navier-Stokes equation, controls the flowfield u. We are often interested in objective functions such as the lift, drag and other forces, which are possible candidates for $\mathcal{F}$. The control parameters might be the geometry of the object itself or perturbations to the boundary conditions.

The *adjoint equation* of the system (2.1) is defined as the linear system

$$\left(\frac{\partial \mathcal{R}}{\partial u}\right)^T \psi = \left(\frac{\partial \mathcal{F}}{\partial u}\right)^T, \tag{2.2}$$

where $\left(\frac{\partial \mathcal{R}}{\partial u}\right)^T$ is the constraint Jacobian, and ψ is the adjoint solution.

The adjoint equation is widely used in analyzing and controlling the system (2.1). For example, many optimization problems, inverse problems and control problems require computing the derivative of the objective function with respect to the control parameters of the original problem (2.1),

$$\frac{d\mathcal{F}(u(\eta),\eta)}{d\eta} = \frac{\partial \mathcal{F}}{\partial \eta} + \frac{\partial \mathcal{F}}{\partial u}\,\frac{du(\eta)}{d\eta},$$

where $u(\eta)$ is an implicit function defined by $\mathcal{R}$, and its derivative $\frac{du(\eta)}{d\eta}$ can be obtained from

$$0 = \frac{d\mathcal{R}(u(\eta),\eta)}{d\eta} = \frac{\partial \mathcal{R}}{\partial \eta} + \frac{\partial \mathcal{R}}{\partial u}\,\frac{du(\eta)}{d\eta}.$$

Therefore, with some manipulation of the adjoint equation (2.2), we get

$$\begin{aligned}\frac{d\mathcal{F}(u(\eta),\eta)}{d\eta} &= \frac{\partial \mathcal{F}}{\partial \eta} - \frac{\partial \mathcal{F}}{\partial u}\left(\frac{\partial \mathcal{R}}{\partial u}\right)^{-1}\frac{\partial \mathcal{R}}{\partial \eta} \\ &= \frac{\partial \mathcal{F}}{\partial \eta} - \psi^T \frac{\partial \mathcal{R}}{\partial \eta},\end{aligned} \tag{2.3}$$

which is a linear function of the adjoint solution.

In this chapter, we focus on the case when the original problem (2.1) is an unsteady problem; e.g., when the constraint $\mathcal{R}$ is a time dependent partial differential equation. In this case, we can order the elements of the state vector and the constraint by time steps, i.e.,

$$u = \left(u^{(1)}, \ldots, u^{(m)}\right)^T \qquad \mathcal{R} = \left(\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(m)}\right)^T$$

where m is the number of time steps, $u^{(i)}$ and $\mathcal{R}^{(i)}$ are the state vector and the constraint at time step i. With this ordering, the matrix $\left(\frac{\partial \mathcal{R}}{\partial u}\right)^T$ in the unsteady adjoint equation (2.2) is well-structured. This is because for unsteady problems, $\mathcal{R}^{(i)}$ only depends on the part of u up to time step i. In other words, a block of the constraint Jacobian $J_{ij} \neq 0$ only if $j \leq i$.

As a result, the Jacobian matrix $\frac{\partial \mathcal{R}}{\partial u}$ is in block-lower-triangular form.

$$\frac{\partial \mathcal{R}}{\partial u} = \begin{bmatrix} J_{1\,1} & & & \\ J_{2\,1} & J_{2\,2} & & \\ \ddots & \ddots & \ddots & \\ J_{m\,1} & \ddots & J_{m\,m-1} & J_{m\,m} \end{bmatrix} \tag{2.4}$$

where each block describes the spatial dependence of the constraint.

$$J_{ts} = \left(\frac{\partial \mathcal{R}^{(t)}}{\partial u^{(s)}} \right).$$

The adjoint equation (2.2) now becomes

$$\begin{bmatrix} J_{1\,1}^T & J_{2\,1}^T & \ddots & J_{m\,1}^T \\ & J_{2\,2}^T & \ddots & \ddots \\ & & \ddots & \ddots \\ & & & J_{m\,m}^T \end{bmatrix} \begin{bmatrix} \psi^{(1)} \\ \psi^{(2)} \\ \vdots \\ \psi^{(m)} \end{bmatrix} = \begin{bmatrix} b^{(1)} \\ b^{(2)} \\ \vdots \\ b^{(m)} \end{bmatrix}, \tag{2.5}$$

where,

$$b^{(t)} = \left(\frac{\partial \mathcal{F}}{\partial u^{(t)}} \right)^T.$$

In this representation, the adjoint solution ψ is split into m parts. We call $\psi^{(t)}$ the adjoint solution at time step t.

Moreover, if $\mathcal{R}$ comes from a discretized unsteady differential equations, the Jacobian matrix is also block-banded. In this case, $\mathcal{R}^{(t)}$ only depends on the part of u that is in a neighborhood of time step t, and $J_{ts} \neq 0$ only if s is in a neighborhood of t. Hence the Jacobian matrix has a block-bandwidth that depends on the temporal discretization scheme. For example, if the original problem uses a one-step scheme, $\mathcal{R}^{(t)}$ depends only on $u^{(t)}$ and $u^{(t-1)}$. In this case, the block-bandwidth is two. If it is a two-step scheme, then the block-bandwidth is three, etc. In particular, if the differential equation is discretized using an explicit scheme, then $\mathcal{R}^{(t)} = u^{(t)} - f(u^{(t-1)}, \ldots)$, and the diagonal blocks of the Jacobian are identity matrices

$$J_{t\,t} = \left(\frac{\partial \mathcal{R}^{(t)}}{\partial u^{(t)}} \right) = I.$$

Note that in this case, the entire Jacobian matrix is lower-triangular instead of just block-lower-triangular.

In addition to the fixed block-bandwidth, each block of the Jacobian matrix is sparse when the original problem is a discretized partial differential equation. This fact follows because in most spatial discretization schemes, the constraint $\mathcal{R}$ at a mesh-point only depends on its neighboring mesh-points. Denote

$$u^{(t)} = \left(u_1^{(t)}, \ldots, u_n^{(t)}\right)^T \qquad \mathcal{R}^{(t)} = \left(\mathcal{R}_1^{(t)}, \ldots, \mathcal{R}_n^{(t)}\right)^T,$$

where $\mathcal{R}_i^{(t)}$ and $u_i^{(t)}$ are the constraint residue and state variable at mesh-point i and time step t. Then $\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_j^{(s)}}$ is nonzero only if j is a neighbor mesh-point of i. Therefore, each block of the constraint Jacobian

$$J_{ts} = \begin{bmatrix} \frac{\partial \mathcal{R}_1^{(t)}}{\partial u_1^{(s)}} & \cdots & \frac{\partial \mathcal{R}_1^{(t)}}{\partial u_n^{(s)}} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{R}_n^{(t)}}{\partial u_1^{(s)}} & \cdots & \frac{\partial \mathcal{R}_n^{(t)}}{\partial u_n^{(s)}} \end{bmatrix}$$

is an $n \times n$ sparse matrix, where n is the number of grid points in space.

Thus far, we have studied the adjoint as the solution of a linear system – a well-structured sparse system if the original problem is a discretized PDE in space and time. One factor that distinguishes the unsteady adjoint equation from other linear systems is its huge size. In an unsteady PDE, the size of the adjoint linear system is $N = n \cdot m$. Where n is proportional to the number of spatial mesh points (up to hundreds of millions), and m is the number of time steps (tens of thousands). For many recently attempted computational fluid dynamic problems, the size of the adjoint equation as a linear system is tens of trillions. As a result, solving adjoint equations requires different strategies and techniques than solving general linear systems.

One direct consequence of such a huge size is the availability of required memory to store the entire linear system. Therefore, it is necessary to use a piece-by-piece approach to solve the adjoint equation. In other words, the solution of the adjoint equation should be computed one piece at a time, based on the information of the primal equation given one piece at a time. For this reason, the matrix ordering is a key consideration in developing an algorithm for solving unsteady adjoint equations. Another consideration is how the adjoint solution will be used. Often applications only need part of the solution, or a linear function

of the solution. Algorithms that directly compute these final quantities without computing the full solution would be more economical. Note that in this case, we generalize the term "*solving adjoint equations*" to computing linear functions of the adjoint solution without computing the full solution.

2.3 Exact solution of the adjoint equations

The structure of the adjoint equation, as shown in equation (2.5), makes the piece-by-piece strategy possible. One approach is to utilize its block-lower-triangular structure and solve using block-back-substitution. In the block-back-substitution, we first solve for $\psi^{(m)}$, then solve for $\psi^{(m-1)}$, and finally solve for $\psi^{(1)}$. This approach for solving the exact solution of an unsteady adjoint equation is consistent with the algorithm of backward automatic differentiation (Griewank, 2003).

This way of solving the unsteady adjoint equation is a time-reverse algorithm. When solving (2.5) using block-back-substitution, the first step is to solve for $\psi^{(m)}$ using block $J^T_{m\,m}$ of the matrix and $b^{(m)}$ of the right-hand side. Since the $J^T_{m\,m}$ and $b^{(m)}$ depend on $u^{(m)}$, the first step of solving the adjoint equation requires the solution of the original problem at the last time step. As the block-backward-substitution continues, the solution of the original problem is required in a time-reverse order. As a result, for this simple method to work efficiently, the solution of the original problem at all time steps, namely the full *trajectory*, must be stored in memory.

However, for even moderately large problems, the memory required to store the full trajectory is too large. One solution to this problem is to use a *checkpointing scheme* (Griewank, 1992) (Griewank, 2003) (Walther & Griewank, 2004). These methods are based on the idea of "divide and conquer." Although the whole trajectory cannot be stored, we can carry out the computation by restarting from a set of checkpoints. Additional forward iterations of the original problem move the solution between checkpoints (Charpentier, 2001).

In 1992 Griewank (Griewank, 1992) first proposed the scheme *Revolve*, which uses this idea to achieve optimal logarithmic behavior in terms of both computational time and memory requirement. Revolve and many other checkpointing schemes proposed since then use $O(\log m)$ times the memory and computational time of the original problem (where m is the number of time steps). Using these schemes, the cost of solving the adjoint equation can

be only a relatively small factor (≈ 3 to 10) times the cost of solving the original problem, even if the number of time steps is large. A new checkpointing scheme will be described in the next chapter.

2.4 Monte Carlo method for the adjoint equation

In this section, we propose a Monte Carlo method for solving the unsteady adjoint equation. Both the memory requirement and computational time of this scheme are $O(1)$ times that of the original problem, independent of the number of time steps of the original problem. As demonstrated in Section 2.7, our Monte Carlo method has better scaling efficiency than the optimal exact solution method, i.e., revolve.

Monte Carlo methods have been shown to be efficient for solving many systems of linear equations, especially when the system is very large and the required precision is relatively low (Tan, 2002) (Dimov, 1998) (Alexandrov, 1998). These methods craft statistical estimators whose mathematical expectation is a component of the solution vector. Random sampling of these estimators yields approximate solutions (Tan, 2002) (Rubinstein, 1981) (Westlake, 1968). The main ideas of these methods were proposed by von Neumann and Ulam, and were extended by Forsythe and Liebler (Forsythe & Liebler, 1950).

Using Monte Carlo methods has several known advantages for solving linear equations. First, the computational cost of obtaining one component of the solution vector using these methods is independent of the size of the linear system (Tan, 2002). More precisely, the computational cost is $O(q\,l)$, where q is the number of random walks and l is their length. Both q and l are independent of the size of the linear system, and can be controlled to obtain any desired precision. Also, Monte Carlo methods are known for their parallel nature. It is often very easy to parallelize them in a coarse-grained manner. Even as early as 1949, Metropolis and Ulam (Metropolisand & Ulam, 1949) noticed the parallelism inherent in this method.

In addition to these advantages, Monte Carlo methods are particularly suitable for solving the unsteady adjoint equation for two reasons. One is that the Monte Carlo method used in this chapter to solve the unsteady adjoint equation is a forward-time procedure. In this procedure, we only use the information necessary to advance to the next time step, which enables us to solve the adjoint equation at the same time as we solve the original equation. Thus we do not need to store the full trajectory, neither do we need to iteratively

resolve the original problem. This is one great advantage over the traditional method. Secondly, we can directly compute the inner product of the solution ψ with a given vector c, without computing ψ. And the cost of computing $c^T\psi$ using this method is only $O(q\,l)$, which is the same as the cost of computing one component of the solution vector. In computing $\frac{d\mathcal{F}}{d\eta}$ using equation (2.3), we can take advantage of this by representing $\psi^T\frac{\partial\mathcal{R}}{\partial\eta}$ as the inner product of ψ with $dim(\eta)$ given vectors

$$\psi^T\frac{\partial\mathcal{R}}{\partial\eta} = \left(\psi^T\frac{\partial\mathcal{R}}{\partial\eta_1}, \psi^T\frac{\partial\mathcal{R}}{\partial\eta_2}, \ldots, \psi^T\frac{\partial\mathcal{R}}{\partial\eta_\xi}\right) \qquad \text{where } \xi = dim(\eta)\ .$$

When the dimension of the control vector $dim(\eta)$ is much smaller than the dimension of the state vector $dim(u)$, we can save a lot of computational cost by directly computing $\psi^T\frac{\partial\mathcal{R}}{\partial\eta}$. This is useful in applications such as wall control for drag reduction (Choi *et al.*, 1993) (Bewley *et al.*, 2001).

In the remainder of this section, Section 2.4.1 introduces the (preconditioned) Neumann series representation of the solution. In Section 2.4.2, we construct the Markovian random walk and the D estimator. We prove that the D estimator is an unbiased estimator to the Neumann series representation of the solution. These discussions are valid for general linear systems and are presented in more detail in (Okten, 2005). Section 2.4.3 describes our Monte Carlo algorithm designed specifically for solving unsteady adjoint equations using the D estimator. Section 2.4.4 discusses the choice of transition probabilities of the Markovian random walk based on the theory of minimum probable error (Dimov, 1991). Section 2.4.5 discusses choice of preconditioner used in Neumann series representation for our Monte Carlo method. In Section 2.4.6 we fully specify our Monte Carlo algorithm used for Burgers' equation.

2.4.1 Neumann series representation

To simplify notation, let

$$\bar{A} = \left(\frac{\partial\mathcal{R}}{\partial u}\right)^T \qquad \text{and} \qquad \bar{b} = \left(\frac{\partial\mathcal{F}}{\partial u}\right)^T$$

in adjoint equation (2.2). The adjoint equation simplifies to

$$\bar{A}\psi = \bar{b}.$$

We multiply both sides by a block-diagonal preconditioner matrix P, which is easy to invert and preserves the block-upper-triangular structure of the Jacobian $\bar{A}$. Denote

$$A = I - P\bar{A} \qquad \text{and} \qquad b = P\bar{b}. \tag{2.6}$$

We note that A has the same block-upper-triangular and block-banded structure of $\bar{A}$. Now the adjoint equation becomes

$$\psi = A\psi + b. \tag{2.7}$$

The solution ψ to the equation above can be expanded in a Neumann series:

$$\psi = b + Ab + A^2b + A^3b + \dots \tag{2.8}$$

The Neumann series converges and (2.8) is valid if and only if the spectral radius of A is less than one. When it converges, the Neumann series (2.8) is the solution of the adjoint equation (2.7). Note that the inner product of ψ with a given vector c can be represented as

$$c^T\psi = c^T(b + Ab + A^2b + A^3b + \dots) = \sum_{k=0}^{\infty} c^T A^k b \; . \tag{2.9}$$

The Monte Carlo method presented and discussed in this chapter is a random sampling of this infinite series by a Markovian random walk.

2.4.2 Markovian random walk and D estimator

For a given vector c, we use Markovian random walks to create random samples of an estimator D introduced in (2.18) whose mathematical expectations are the inner product of the solution ψ with the given vector c, i.e., $\mathbf{E}(D) = c^T\psi$. In particular, when $c = e_i$, $\mathbf{E}(D) = \psi^T e_i = \psi_i$, and we compute a component of the solution vector. The Markovian random walks have a finite state space with size $N + 1$, where N is the size of the adjoint equation. If we label the states as $1, 2, \dots, N + 1$, each of the first N states correspond to a component of the adjoint equation. The special state, a final exit state, is labeled $N + 1$. The random walks begin from a birth probability r, and follow a transition probability p, where $p(i, j)$ is the transition probability from state i to state j. r and p must satisfy the

following five conditions (Okten, 2005):

$$0 \leq r(j), p(i,j) \leq 1 \text{ for all } i,j, \tag{2.10}$$

$$\sum_{j=1}^{N+1} p(i,j) = 1 \text{ for all } i, \tag{2.11}$$

$$\sum_{i=1}^{N} r(i) = 1, \tag{2.12}$$

$$p(N+1, N+1) = 1 \tag{2.13}$$

$$p(i,j) \neq 0 \Longleftrightarrow A_{i\,j} \neq 0 \text{ and } r(i) \neq 0 \Longleftrightarrow c_i \neq 0, \tag{2.14}$$

where $A_{i\,j}$ is the i,jth entry of the matrix A, and c_i is the ith component of the vector c. In the five conditions above, the first three define r and p as birth and transition probabilities. The Condition (2.13) defines the state $N+1$ as the final exit state. We note that the probability that the Markovian random walk transits from state j to the final exit state is $p(j, N+1) = 1 - \sum_{i=1}^{N} p(i,j)$. Once it reaches the final exit state, it always stays there with probability 1. When this happens, we say that the random walk was *absorbed* in state j. The Condition (2.14) means that the random walks always remain tied to the matrix, which allows us to go from the random walk model described by p and a to the Neumann series (2.9) that involves $A_{i\,j}$ and c_i.

Now we'll relate the random walk to the components of the matrix A. Let the Markov chain be

$$\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_n, \ldots),$$

and

$$\mathbf{i} = (i_0, i_1, \ldots, i_l, N+1, N+1, \ldots), \quad 1 \leq i_j \leq N$$

be a typical path of the Markov chain that begins at state i_0 and is absorbed at state i_l. The probability that the Markov chain takes this path is

$$\mathcal{P}(\alpha = \mathbf{i}) = r(i_0)\, p(i_0 i_1)\, p(i_1 i_2)\ \ldots\ p(i_{l-1} I_l)\, p(i_l, N+1)\,.$$

We define

$$w_{i\,j} = \begin{cases} \dfrac{A_{i\,j}}{p(i,j)} & \text{if } p(i,j) \neq 0 \\ 0 & \text{if } p(i,j) = 0 \end{cases}$$

and as in (Okten, 2005), we define the weight W_k as a random variables on the space of random walks α:

$$W_k(\alpha) = \frac{c_{\alpha_0}}{r(\alpha_0)} \prod_{j=1}^{k} w_{\alpha_j \alpha_{j-1}} \qquad 0 \leq k \leq l \; . \tag{2.15}$$

The following proposition provides insight on why we define the weight in this way.

Proposition 2.4.1. *Assume Conditions (2.10) to (2.14) are satisfied. Denote $(\cdot)_j$ as the jth component of a vector, and $(\cdot)_{ij}$ as the i,j entry of a matrix. Then*

$$\mathbf{E}(W_k I_{\{\alpha_k = j\}}) = \left(c^T A^k\right)_j \; . \tag{2.16}$$

In particular, if $c = e_i$,

$$\mathbf{E}(W_k I_{\{\alpha_k = j\}}) = \left(A^k\right)_{ij} \; . \tag{2.17}$$

Proof. Since $W_0 = \frac{c_{\alpha_0}}{r(\alpha_0)}$ and $\mathcal{P}(\alpha_0 = j) = r(j)$,

$$\mathbf{E}[W_0 I_{\{\alpha_0 = j\}}] = \mathcal{P}(\alpha_0 = j) \frac{c_j}{r(j)} = c_j \; .$$

So (2.16) holds for $k = 0$. Assume it holds for certain k, we prove it holds for $k+1$.

$$\begin{aligned} \mathbf{E}[W_{k+1} I_{\{\alpha_{k+1} = j\}}] &= \mathbf{E}\left[\sum_i \left(I_{\{\alpha_k = i, \alpha_{k+1} = j\}} W_k \; w(i,j)\right)\right] \\ &= \sum_i \mathbf{E}\left[I_{\{\alpha_k = i, \alpha_{k+1} = j\}} W_k\right] \frac{A_{i\,j}}{p(i,j)} \end{aligned}$$

By using the tower property of conditional expectations, "taking out what is known," and applying the Markov property, we have

$$\mathbf{E}\left[I_{\{\alpha_k = i, \alpha_{k+1} = j\}} W_k\right] = \mathbf{E}\left[I_{\{\alpha_k = i\}} W_k\right] p(i,j) \; .$$

Therefore, by the induction hypothesis,

$$\begin{aligned}\mathbf{E}[W_{k+1} I_{\{\alpha_{k+1}=j\}}] &= \sum_i \mathbf{E}\left[I_{\{\alpha_k=i\}} W_k\right] A_{i\,j} \\ &= \sum_i \left(c^T A^k\right)_i A_{i\,j} \\ &= \left(c^T A^{k+1}\right)_j \ .\end{aligned}$$

Thus we conclude that Equation (2.16) holds true for all $k \geq 0$. Equation (2.17) follows trivially. □

This tells us that $W_k I_{\{\alpha_k=j\}}$ is in fact a randomly sparsified version of vector $c^T A^k$. The randomly sparsified vector contains only one nonzero entry. In every step of the random walk, $W_k I_{\{\alpha_k=j\}}$ is multiplied by A, and further sparsified. We can think of it as the sparsified version of $c^T A^k$ multiplied by A, by which we get an approximation of $c^T A^{k+1}$, and then sparsify it. Because the Neumann series (2.9) gives us a relationship between $c^T \psi$ and $c^T A^k$, we can use this relationship to build an estimator for $c^T \psi$ in terms of $W_k I_{\{\alpha_k=j\}}$, which is a randomly sparsified version of $c^T A^k$.

Definition 2.4.2. Define the D estimator by

$$D(\alpha) = \sum_{k=0}^{\infty} W_k(\alpha) b_{\alpha_k} \ , \tag{2.18}$$

where $\alpha = (\alpha_0, \alpha_1, \ldots)$ is the random walk; $(b_1, \ldots, b_N)^T$ is the right-hand side of equation (2.7), and $b_{N+1} = 0$.

Now we prove the main theorem that supports the Monte Carlo method.

Theorem 2.4.3. *Assume that the Neumann series (2.8) converges for $|A|$, and Conditions (2.10) to (2.14) are satisfied. Then the expectation of the D estimator is*

$$\mathbf{E}(D(\alpha)) = c^T \psi \ ,$$

where ψ is the solution of equation (2.7).

Proof. Since $b_{\alpha_k} = \sum_j I_{\{\alpha_k=j\}}$, the expectation of the D estimator can be represented as

$$\begin{aligned}\mathbf{E}[D] &= \mathbf{E}\left[\sum_{k=0}^{\infty} W_k\, b_{\alpha_k}\right] \\ &= \mathbf{E}\left[\sum_{k=0}^{\infty}\sum_j W_k I_{\{\alpha_k=j\}} b_j\right] \\ &= \sum_{k=0}^{\infty}\sum_j \mathbf{E}\left[W_k I_{\{\alpha_k=j\}}\right] b_j\ .\end{aligned}$$

The condition that the Neumann series converges for $|A|$ justifies the exchange of infinite sum and expectation by the dominated convergence theorem. It follows from proposition (2.4.1) that

$$\mathbf{E}[D] = \sum_{k=0}^{\infty}\sum_j (c^T A^k)_j b_j = \sum_{k=0}^{\infty} c^T A^k b$$

which is the Neumann series expansion (2.9). Thus we have

$$\mathbf{E}[D] = c^T\psi \tag{2.19}$$

i.e., D is an unbiased estimator of $c^T\psi$. □

This theorem suggests that we can use the Monte Carlo method based on the D estimator to approximate $c^T\psi$

$$c^T\psi \approx \frac{1}{q}\sum_{p=1}^{q} D(\alpha[p]),$$

where $\alpha[p]$, $1 \le p \le q$ are independent identically distributed random walks. And the following corollary is a direct consequence of Theorem 2.4.3 and the strong law of large numbers.

Corollary 2.4.4. *Under the conditions of Theorem 2.4.3, the estimated solution by the Monte Carlo method*

$$\frac{1}{q}\sum_{p=1}^{q} D(\alpha[p]) \to c^T\phi$$

almost surely as $q \to \infty$.

This result justifies our Monte Carlo approach for solving adjoint equations by stating

that as the number of random walks increases, the solution of the Monte Carlo method asymptotically converges to the exact solution.

2.4.3 Monte Carlo algorithm

In this section, we describe the Monte Carlo algorithm for solving the unsteady adjoint equation based on the D estimator constructed in the previous section.

We note that Condition (2.14) of the transition probability matrix guarantees that $\mathbf{P}$ has the same block-upper-triangular and block-banded structure as matrix A. We remember that the components of u are ordered by time steps. As a result, random walks defined by this transition probability matrix can only possibly walk to later time steps (upper-diagonal blocks of $\mathbf{P}$ are nonzero), or walk within the same time step (diagonal blocks of $\mathbf{P}$ are nonzero), but never walk backward to previous time steps (lower-diagonal blocks of $\mathbf{P}$ are always zero). Therefore, the Markovian random walks only go forward in time. This makes the following forward-time Monte Carlo algorithm possible.

In this algorithm, we generate q independent identically distributed random walks $\alpha[p]$, $1 \leq p \leq q$. Each of them has transition probabilities $p(i, j)$ and birth probabilities $r(i)$ that satisfies the Conditions (2.10) to (2.14). We will discuss the choices for r and p in the next section. We denote $\alpha_k[p]$ as the current position of random walk $\alpha[p]$, and we say that a random walk $\alpha[p]$ is at time step t if $\alpha_k[p]$ is in the range of indices that represent time step t. Let $W[p]$ denote $W_k(\alpha[p])$, the weight of random walk at current step. $D[p]$ stores the accumulative sum of the D estimator (2.18), which equals to $D(\alpha)$ after the random walk is absorbed.

1. For each $1 \leq p \leq q$, choose $\alpha_0[p]$ randomly by birth probability vector r, and initialize
$$W[p] = c_{\alpha_0[p]}/r(\alpha_0[p]) \; ; \quad D[p] = W[p]b_{\alpha_0[p]}.$$
Then start from $t = 1$, do steps 2 to 4, until completing the last time step $t = m$.

2. Solve the original problem at time step t, which enables us to compute the corresponding blocks of matrix A and p.

3. For each random walk $\alpha_[p]$ that is at time step t, choose its next state $\alpha_{k+1}[p]$ randomly by transition probabilities p. If $\alpha_{k+1}[p]$ is not the final exit state, update
$$W[p] = W[p]w_{\alpha_k[p]\alpha_{k+1}[p]} \; ; \quad D[p] = D[p] + W[p]b_{\alpha_{k+1}[p]}.$$
If $\alpha_{k+1}[p]'$ is the final exit state, the random walk is absorbed and we freeze $D[p]$.

4. Repeat step 3 until all random walks at time step t are either absorbed or left the time step. If $t < m$, then let $t = t + 1$ and go to step 2.

5. After completing the last time step m, all random walks are absorbed. Compute the sample mean of the estimators $\frac{1}{q}\sum_{p=1}^{q} D[p]$, which is our approximation to $c^T\psi$.

It is clear that this is a forward-time algorithm in which we only need to store the current time steps of the original problem and no iterations are needed. Further, it directly yields the inner product of the solution with the given vector. Indeed, there is no difficulty with this algorithm to solve multiple linear functions of the solution vector at the same time. This property is useful in many adjoint-based methods. For example, in computing $\frac{d\mathcal{F}(u(\eta),\eta)}{d\eta}$ using formula (2.3), we can use the Monte Carlo method to directly compute $\psi^T\frac{\partial\mathcal{R}}{\partial\eta}$, which is the inner product of the adjoint solution ψ with $\frac{\partial\mathcal{R}}{\partial\eta_i}, i = 1, ..., dim(\eta)$. This can be directly obtained from the algorithm described above. The cost of this algorithm is $O(\xi\, q\, l)$ plus the cost of the original problem, where ξ is the dimension of the control

vector; q is the number of random walks for each evaluation of inner product; and l is the average length of the random walk, which is proportional to the number of time steps m of the original problem. When the dimension of the control vector is much smaller than the mesh size, and the required precision is relatively low (which allows us to choose a small q), the cost of solving the adjoint equation in this algorithm is a small overhead. This is particularly attractive especially compared to solving the exact solution of the unsteady adjoint equation, which is significantly more costly in computation time and storage than the original problem.

2.5 Analysis of the Monte Carlo method

In the previous section, we derived the algorithm of using random walk Monte Carlo to solve the discrete adjoint equation, and theoretically proved that as the number of samples increases, the solution obtained by our method converges asymptotically to the exact solution. In practice, however, it is only possible to run a finite number of random walks with limited computational resources. In this section, we address the following questions: How much error is made by running a finite number of random walks, and how this error can be controlled and minimized?

Before we start, we define the probable error in order to quantify the difference between the Monte Carlo approximation and the exact solution of the adjoint equation.

Definition 2.5.1. Let I be the value to be estimated by Monte Carlo method, and D be its unbiased estimator. The probable error for the Monte Carlo method is defined to be

$$r = \sup\left\{s : \mathcal{P}\left(|I - D| \geq s\right) > \frac{1}{2}\right\} \tag{2.20}$$

The probable error specifies the range which contains 50% of the possible values of the estimator. In the case of continuous distribution, this is equivalent to the definition in (Tan & Alexandrov, 2001) and (Sobol, 1973).

The probable error is closely related to the variance of the estimator. Suppose $D_1, \ldots, D_q$ are independent and identically distributed samples of D, If the variance of estimator D is bounded, the Central Limit Theorem

$$\mathcal{P}\left(\frac{\sum D_i}{q} - I \leq x\sqrt{\frac{\mathrm{V}arD}{q}}\right) \to \Phi(x)$$

holds. When q is large, the error of the average of the q samples, defined as

$$P\left(\left|\frac{\sum D_i}{q} - I\right| \leq r\right) = 0.5\ ,$$

is (Tan, 2002) (Tan & Alexandrov, 2001)

$$r \approx 0.6745\left(\frac{\mathrm{Var}D}{q}\right)^{1/2}. \tag{2.21}$$

Therefore, the probable error decreases when the number of samples q increases, or when the variance of the estimator D decreases. Using this formula, we can estimate and control the probable error of our Monte Carlo method by estimating the variance of the D estimator.

In the remainder of this section, we focus on the variance of the D estimator. To make the mathematical derivation cleaner, we denote

$$p_{ij} = p(i, j)$$

to be the transition probability from state i to state j,

$$p_i = p(i, N+1)$$

to be the transition probability from state i to the final exit state, and

$$r_i = r(i)$$

to be the birth probability at state i.

2.5.1 Variance decomposition

Suppose the mean and variance of the D estimator of a random walk starting from state i are $\mathbf{E}_i(D)$ and $\mathbf{V}_i(D)$, respectively. Since the Markov chain at state i has p_i probability of going to the final exit state, and p_{ij} probability of going to the jth state, we know

$$\mathbf{E}_i(D) = p_i\left(\frac{1}{p_i}\, b_i\right) + \sum_{j:\, A_{ij}\neq 0} p_{ij}\left(\frac{A_{ij}}{p_{ij}}\,\mathbf{E}_j(D)\right) = b_i + \sum_{j:\, A_{ij}\neq 0} A_{ij}\,\mathbf{E}_j(D)$$

which corresponds to the linear equation (2.7) we want to solve. Apply the same analysis to the expectation of D^2, we can obtain a formula for the variance of the D estimator,

$$\begin{aligned}\mathbf{V}_i(D) &= \mathbf{E}_i\left(D^2\right) - \mathbf{E}_i(D)^2 \\ &= p_i\left(\frac{b_i}{p_i}\right)^2 + \sum_{j:\, A_{ij}\neq 0} p_{ij}\left(\frac{A_{ij}^2}{p_{ij}^2}\,\mathbf{E}_j(D^2)\right) - \mathbf{E}_i(D)^2 \\ &= \left(\frac{b_i^2}{p_i} + \sum_{j:\, A_{ij}\neq 0}\frac{A_{ij}^2}{p_{ij}}\,\mathbf{E}_j(D)^2 - \mathbf{E}_i(D)^2\right) + \left(\sum_{j:\, A_{ij}\neq 0}\frac{A_{ij}^2}{p_{ij}}\,\mathbf{V}_j(D)\right)\end{aligned} \tag{2.22}$$

We can see that the variance of a random walk starting at state i comes from two parts; the first part is

$$\mathbf{V}_i^{(1)}(D) = \frac{b_i^2}{p_i} + \sum_{j:\, A_{ij}\neq 0}\frac{A_{ij}^2}{p_{ij}}\,\mathbf{E}_j(D)^2 - \mathbf{E}_i(D)^2\ . \tag{2.23}$$

This part of variance is caused by the first step of the random walk. The second part,

$$\mathbf{V}_i^{(2)}(D) = \sum_{j:\, A_{ij}\neq 0}\frac{A_{ij}^2}{p_{ij}}\,\mathbf{V}_j(D)\ , \tag{2.24}$$

is a weighted average of the variance of random walks starting from all the states that state i leads to. This part of variance is caused solely by the random walk starting from the second step.

Similarly, we can calculate the variance of the D estimator of a random walk starting with birth probability r_i,

$$\begin{aligned}\mathbf{V}_r(D) &= \mathbf{E}_r\left(D^2\right) - \mathbf{E}_r(D)^2 \\ &= \sum_{i\,:\,r_i\neq 0} r_i\left(\frac{c_i^2}{r_i^2}\,\mathbf{E}_i\left(D^2\right)\right) - \mathbf{E}_r(D)^2 \\ &= \left(\sum_{i\,:\,r_i\neq 0}\frac{c_i^2}{r_i}\,\mathbf{E}_i(D)^2 - \mathbf{E}_r(D)^2\right) + \left(\sum_{i\,:\,r_i\neq 0}\frac{c_i^2}{r_i}\,\mathbf{V}_i\left(D^2\right)\right)\end{aligned} \tag{2.25}$$

where $\mathbf{E}_i(D)$ and $\mathbf{V}_i(D)$ are the mean and variance of the random walk starting deterministically from the state i. We can see that the variance of a random walk starting with birth

probability r_i also comes from two parts. The first part,

$$\mathbf{V}_r^{(1)}(D) = \sum_{i\,:\,r_i \neq 0} \frac{c_i^2}{r_i}\,\mathbf{E}_i(D)^2 - \mathbf{E}_r(D)^2\,, \tag{2.26}$$

is caused by the randomness of the birth state. The second part,

$$\mathbf{V}_r^{(2)}(D) = \sum_{i\,:\,r_i \neq 0} \frac{c_i^2}{r_i}\,\mathbf{V}_i\left(D^2\right), \tag{2.27}$$

is a weighted average of the variance of random walks starting from all the possible birth states. This part of variance is caused by the random walk starting from each possible birth state.

Based on this split of variance, the next subsection discusses choice of transition and birth probabilities p_i and r_i that reduces each component of the decomposition.

2.5.2 Choice of transition and birth probabilities

Finding the probabilities that make the variance as small as theoretically possible has been shown to be impractically time-consuming for Monte Carlo linear solvers (Dimov, 1991) (Dimov & Tonev, 1993). For this reason, we focus on finding "almost optimal" transition and birth probabilities by minimizing upper bounds of the variances.

The upper bounds on which we base our almost optimal transition probabilities are

$$\begin{aligned}\mathbf{V}_i^{(1)}(D) &= \frac{b_i^2}{p_i} + \left(\sum_{j\,:\,A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}}\,\mathbf{E}_j(D)^2\right) - \mathbf{E}_i(D)^2 \\ &\leq \frac{b_i^2}{p_i} + \left(\sum_{j\,:\,A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}}\right)\mathbf{B}^2 - \mathbf{E}_i(D)^2\end{aligned}$$

and

$$\begin{aligned}\mathbf{V}_i^{(2)}(D) &= \sum_{j\,:\,A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}}\,\mathbf{V}_j(D) \\ &\leq \left(\sum_{j\,:\,A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}}\right)\max_{j\,:\,A_{ij}\neq 0}\mathbf{V}_j(D),\end{aligned}$$

where

$$\mathbf{B} = \max |\mathbf{E}_j(D)| \ .$$

These bounds are chosen because individual $\mathbf{E}_j(D)^2$ and $\mathbf{V}_j(D)$ are not known *a priori*. They are therefore substituted by a common upper bound. The optimal p_{ij} and p_i for these upper bounds, under the constraints

$$\sum_j p_{ij} + p_i = 1 \quad \text{and} \quad p_{ij} \geq 0,$$

are given by the formulae [1]

$$p^*_{ij} = \frac{|A_{ij}|\,\mathbf{B}}{|b_i| + \sum_j |A_{ij}|\,\mathbf{B}} \tag{2.28}$$

and

$$p^*_i = \frac{|b_i|}{|b_i| + \sum_j |A_{ij}|\,\mathbf{B}}. \tag{2.29}$$

These are the almost optimal transition probabilities. Note that since $\mathbf{B}$ is not known *a priori*, it needs to be estimated unless $b_i = 0$.

Similarly, we construct upper bounds for $\mathbf{V}^{(1)}_r(D)$ and $\mathbf{V}^{(2)}_r(D)$:

$$\begin{aligned} \mathbf{V}^{(1)}_r(D) &= \left(\sum_{i\,:\,r_i \neq 0} \frac{c_i^2}{r_i}\, \mathbf{E}_i(D)^2 \right) - \mathbf{E}_r(D)^2 \\ &\leq \left(\sum_{i\,:\,r_i \neq 0} \frac{c_i^2}{r_i} \right) \mathbf{B}^2 - \mathbf{E}_r(D)^2 \end{aligned} \ ,$$

$$\begin{aligned} \mathbf{V}^{(2)}_r(D) &= \sum_{i\,:\,r_i \neq 0} \frac{c_i^2}{r_i}\, \mathbf{V}_i\left(D^2\right) \\ &\leq \left(\sum_{i\,:\,r_i \neq 0} \frac{c_i^2}{r_i} \right) \left(\max \mathbf{V}_i\left(D^2\right) \right) \end{aligned} \ .$$

The almost optimal birth probabilities that minimizes these upper bounds are given by the formula

$$r^*_i = \frac{|c_i|}{\sum_i |c_i|} \ . \tag{2.30}$$

[1] (2.28) and (2.29) together minimizes upper bound of $\mathbf{V}^{(1)}_i(D)$; (2.28) alone minimizes upper bound of $\mathbf{V}^{(2)}_i(D)$ under the constraints.

Because the almost optimal transition and birth probabilities minimize upper bounds of the estimator variance, we use this choice of probabilities in all numerical experiments presented later in this chapter.

2.5.3 Growth of variance

This subsection studies how the variance of our D estimator can grow, assuming the almost optimal transition and birth probabilities derived in the previous subsection. First, we incorporate the optimal probabilities into the upper bounds of $\mathbf{V}_i^{(1)}$ and $\mathbf{V}_i^{(2)}$, we get

$$\mathbf{V}_i^{(1)}(D) \leq (|b_i| + \mathbf{\Gamma}_i \mathbf{B})^2 - \mathbf{E}_i(D)^2,$$

and

$$\mathbf{V}_i^{(2)}(D) \leq \left(\frac{|b_i|}{\mathbf{B}} + \mathbf{\Gamma}_i\right) \mathbf{\Gamma}_i \max_{j:\, A_{ij} \neq 0} \mathbf{V}_j(D),$$

where

$$\mathbf{\Gamma}_i = \sum_j |A_{ij}|.$$

The total variance is the sum of the two components of the variance decomposition, thus

$$\mathbf{V}_i(D) \leq \left((|b_i| + \mathbf{\Gamma}_i \mathbf{B})^2 - \mathbf{E}_i(D)^2\right) + \left(\frac{|b_i|}{\mathbf{B}}\mathbf{\Gamma}_i + \mathbf{\Gamma}_i^2\right) \max_{j:\, A_{ij} \neq 0} \mathbf{V}_j(D). \tag{2.31}$$

This equation characterizes the growth of variance as the Monte Carlo random walk proceeds. In the case of explicit time-stepping, all j such that $A_{ij} \neq 0$ are in the next time step of i. As the random walk proceeds through time steps, the variance can suffer from exponential growth if the multiplicative factor $\left(\frac{|b_i|}{\mathbf{B}}\mathbf{\Gamma}_i + \mathbf{\Gamma}_i^2\right)$ is greater than 1. If this is the case, the probable error can be too large for our Monte Carlo method to be practical. On the other hand, if this factor is less or equal to 1, the variance grows at most linearly. For this reason, the size of multiplicative factor is critical in the efficiency of our Monte Carlo method.

In the case of conservation law PDEs discretized with a positive coefficient scheme, we know the size of this multiplicative factor. The discrete conservation property guarantees that

$$\sum_j A_{ij} = 1,$$

for all but boundary grid points. Also, all A_{ij} are non-negative in a positive coefficient scheme. As a result of these two properties,

$$\Gamma_i = \sum_j |A_{ij}| = \sum_j A_{ij} = 1. \tag{2.32}$$

In addition, if the adjoint equation does not have a source term, then $b_i = 0$, and the multiplicative factor

$$\frac{|b_i|}{\mathbf{B}} \mathbf{\Gamma}_i + \mathbf{\Gamma}_i^2 = 1.$$

This implies that the probable error of our Monte Carlo method does not increase exponentially in this case. This is indeed true, as seen in our numerical experiments, that the error of our Monte Carlo adjoint solver decreases as the number of time steps increases. In case the adjoint equations have a source term, the discretized source term b_i is of order of Δt, thus the multiplicative factor

$$\frac{|b_i|}{\mathbf{B}} \mathbf{\Gamma}_i + \mathbf{\Gamma}_i^2 = 1 + O(\Delta t).$$

This suggests that for a fixed T, as the discretization refines, the total amount of variance growth remains bounded.

For systems other than scalar transport equations, equation (2.32) may not be true. In many cases, a proper choice of preconditioner is required to prevent exponential growth of variance.

2.5.4 Choice of preconditioner

The choice of the preconditioner matrix P in equation (2.6) [2] controls the behavior of the Neumann series, and influences the variance growth of the estimator D. A good choice can improve the precision of the result and reduce the cost by requiring fewer samples, while a bad choice can make the variance grow exponentially. Thus, the main purpose of the preconditioner is to control the multiplicative factor in the variance growth by reducing Γ_i. Just like choosing a preconditioner a linear system, there is no universal best choice. Still, there are a few preconditioners that we wish to mention in relation to unsteady adjoint equations.

[2] Not to be confused with the transition probability matrix $\mathbf{P}$.

First, if the Jacobian matrix is diagonally dominant, a possible choice of preconditioner is the inverse of the diagonal part of the Jacobian. This method is called *diagonal splitting*. Diagonal splitting makes the Neumann series equivalent to the Jacobian iteration scheme, which has guaranteed convergence for diagonal dominant matrices.

Tan proposes a relaxed Monte Carlo linear solver in (Tan & Alexandrov, 2001) (Tan, 2002), which is equivalent to choosing a diagonal preconditioner that is not equal to the diagonal part of the Jacobian matrix. It was shown that this approach has improved performance over diagonal splitting for many problems.

Srinivasan (Srinivasan, 2003) studied non-diagonal splitting Monte Carlo solvers, which is equivalent to choosing the preconditioner to be inverse of the diagonal and first sub-diagonal or super-diagonal of $\bar{A}$. His approach is more suitable for a larger class of problems than diagonal conditioning. We have not yet investigated the last two preconditioners in the context of solving unsteady adjoint equations.

In the context of partial differential equations, choosing a preconditioner that transforms the random walk into the frequency space is currently being investigated. Preliminary results show that this preconditioner may be a solution to certain problems on which achieving a bounded variance growth using existing preconditioners is difficult.

2.6 Example of Monte Carlo algorithm: Burgers' equation

In this section, we demonstrate a concrete example of the Monte Carlo algorithm for solving adjoint equations. The example is Burgers' equation,

$$\mathcal{R}(x,t) = u_t + \left(\frac{u^2}{2}\right)_x = 0,$$

discretized temporally by the forward Euler scheme, and spatially by the first-order upwinding scheme. Our original problem is the fully discretized equation

$$\mathcal{R}_i^{(t)} = u_i^{(t)} - u_i^{(t-1)} + \frac{\Delta t}{\Delta x}\left(f_{i+\frac{1}{2}}^{(t-1)} - f_{i-\frac{1}{2}}^{(t-1)}\right) = 0, \qquad t = 1, 2, \ldots, m, \tag{2.33}$$

where f is the numerical flux computed using the up-winding formula

$$f_{i+\frac{1}{2}}^{(t)} = \begin{cases} \frac{1}{2}\left(u_i^{(t)}\right)^2 & \text{if } u_i^{(t)} + u_{i+1}^{(t)} > 0 \\ \frac{1}{2}\left(u_{i+1}^{(t)}\right)^2 & \text{if } u_i^{(t)} + u_{i+1}^{(t)} \leq 0. \end{cases}$$

In this case, the full state vector u is

$$u = \left(u_1^{(1)}, \ldots, u_n^{(1)}, \quad u_1^{(2)}, \ldots, u_n^{(2)}, \quad \ldots\ldots, \quad u_1^{(m)}, \ldots, u_n^{(m)}\right)^T$$

and the full constraint is

$$\mathcal{R} = \left(\mathcal{R}_1^{(1)}, \ldots, \mathcal{R}_n^{(1)}, \quad \mathcal{R}_1^{(2)}, \ldots, \mathcal{R}_n^{(2)}, \quad \ldots\ldots, \quad \mathcal{R}_1^{(m)}, \ldots, \mathcal{R}_n^{(m)}\right)^T$$

where n is the number of mesh points, and m is the number of time steps. The size of the adjoint equation as a linear system is $N = mn$.

For this example, we'll analyze the structure of the adjoint equation, derive the birth probability matrix and transition probability matrix, and walk through the Monte Carlo algorithm. Let's begin with the constraint Jacobian matrix. The forward Euler temporal discretization scheme is a one-step scheme, and the constraint (2.33) at a time step t only depends on u at time step t and $t-1$. Hence, the only nonzero blocks in the Jacobian (2.4) are J_{tt} and $J_{t\,t-1}$, $t = 1, \ldots, m$. Moreover, the equation is discretized using an explicit scheme, so $\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_j^{(t)}}$ is nonzero only if $i = j$. Therefore, the diagonal blocks of the Jacobian are identity matrices,

$$J_{tt} = \frac{\partial \mathcal{R}^{(t)}}{\partial u^{(t)}} = I.$$

In this case, we use no preconditioner, and

$$A = I - \left(\frac{\partial \mathcal{R}}{\partial u}\right)^T = \begin{bmatrix} 0 & -J_{21}^T & & \\ & 0 & \ddots & \\ & & \ddots & -J_{m\,m-1}^T \\ & & & 0 \end{bmatrix}.$$

The Neumann series associated with this matrix has finite length m.

Each block $J_{t\,t-1}^T$ in this matrix is well-structured. We note that the residue (2.33) at

mesh-point i only depends on u at mesh-point $i-1, i$ and $i+1$. Thus $\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_j^{(t)}}$ is nonzero only if $i-1 \leq j \leq i+1$, and the off-diagonal blocks $J_{t\,t-1}$ are tri-diagonal matrices with entries

$$\begin{aligned}
\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_{i-1}^{(t-1)}} &= -\,a_i^{(t)} \frac{\Delta t}{\Delta x} u_{i-1}^{(t)} \\
\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_i^{(t-1)}} &= -\,1 - b_i^{(t)} \frac{\Delta t}{\Delta x} u_i^{(t)} \\
\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_{i+1}^{(t-1)}} &= -\,c_i^{(t)} \frac{\Delta t}{\Delta x} u_{i-1}^{(t)}
\end{aligned} \tag{2.34}$$

where

$$a_i^{(t)} = \begin{cases} -1 & \text{if } u_{i-1}^{(t)} + u_i^{(t)} > 0 \\ 0 & \text{if } u_{i-1}^{(t)} + u_i^{(t)} \leq 0 \end{cases}$$

$$b_i^{(t)} = \begin{cases} 1 & \text{if } u_{i-1}^{(t)} + u_i^{(t)} > 0 \text{ and } u_i^{(t)} + u_{i+1}^{(t)} > 0 \\ -1 & \text{if } u_{i-1}^{(t)} + u_i^{(t)} \leq 0 \text{ and } u_i^{(t)} + u_{i+1}^{(t)} \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$c_i^{(t)} = \begin{cases} 1 & \text{if } u_i^{(t)} + u_{i+1}^{(t)} \leq 0 \\ 0 & \text{if } u_i^{(t)} + u_{i+1}^{(t)} > 0 \end{cases} .$$

We compute the transition probabilities using (2.28), choosing the absorption probability to be 0 at $t < m$, and 1 at $t = m$. We get the transition probability matrix

$$\mathbf{P} = \begin{bmatrix} 0 & P_1 & & & 0 \\ & 0 & \ddots & & 0 \\ & & \ddots & P_{m-1} & 0 \\ & & & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

where the last row and column correspond to the final exit state, and each P_t is an $n \times n$ tri-diagonal matrix. We note that following this transition probability matrix, each step of a random walk goes one time step forward. At the next step, a random walk either stays at the same mesh-point, or goes to its left or right neighbor, with probabilities specified by P_t

at the current time step. Therefore, in our Monte Carlo method, the spatial distribution of the random walks follows transition probabilities P_t at each time step, and all random walks are absorbed at the mth time step. This is consistent to the finite length of a Neumann series associated with the matrix A.

Assume that we are interested in solving for the adjoint solution at the first time step $\psi^{(1)}$, we can approximate it by starting q random walks from each of the n mesh-points at the first time step. Denote $\alpha[p,i]$ as the pth random walk starting at mesh-point i, and $\alpha_t[p,i]$ its position at time step t. The Monte Carlo algorithm in this case is

1. For each $1 \leq p \leq q, 1 \leq i \leq n$, choose $\alpha_0[p,i] = x$ and initialize

$$W[p,i] = 1\ ; \quad D[p,i] = b_i.$$

   Then start from $t = 1$, do steps 2 to 4, until completing the last time step $t = m$.

2. Solve the original problem at time step t for $u^{(t)}$. Compute tri-diagonal matrix $-J^T_{t+1\,t}$ using (2.34), and transition probabilities P_t at this time step using (2.28).

3. For each random walk, choose its next state $\alpha_{t+1}[p]$ according to P_t. Update

$$W[p,i] = W[p,i] w_{\alpha_t[p,i]\alpha_{t+1}[p,i]}\ ; \quad D[p,i] = D[p,i] + W[p,i] b_{\alpha_{t+1}[p,i]}.$$

4. Let $t = t + 1$, if $t < m$, go to step 2.

5. At last time step m, all random walks are absorbed. Compute the sample mean of the estimators $\frac{1}{q}\sum_{p=1}^{q} D[p,i]$ for each i, which is our approximation to $\psi_i^{(1)}$.

2.7 Numerical experiments

In this section, we use three numerical experiments to demonstrate the convergence property and scaling efficiency of our Monte Carlo adjoint solver, as well as its performance in solving

an inverse problem. These experiments are done with Burgers' equation, discretized with numerical scheme (2.33). Because we used an explicit temporal discretization scheme for the original problem, we use no preconditioning in the Monte Carlo algorithm (see section 4.4). The transition probabilities and birth probabilities are chosen based on (2.28) and (2.30). The absorption probabilities are chosen to be 1 at the last time step, and 0 at other time steps except at boundaries. All experiments are done on a desktop with two Intel(R) Xeon(TM) 3.00GHz CPUs, 2GB memory, GNU/Linux 2.6.9-42.0.10.ELsmp.

2.7.1 Convergence of Monte Carlo adjoint solver

This experiment demonstrates that the Monte Carlo adjoint solution converges to the exact solution as the number of random walks increases. We solve the Burgers' equation with initial condition

$$u(x,0) = \sin(\pi x), \qquad x \in [0,1] \tag{2.35}$$

in the time interval $[0, 0.25]$. First-order up-winding finite volume scheme is used on 100 grid points uniformly spaced in $[0,1]$; the CFL number for time integration is set to 0.5. On the other hand, the adjoint equation is solved with final condition

$$\phi(x,0.25) = \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \qquad x \in [0,1] \tag{2.36}$$

in the same time interval $[0, 0.25]$, where $\mu = 0.5$ and $\sigma = 0.2$. Both Burgers' equation and the adjoint equation use zero Dirichlet boundary conditions.

Our Monte Carlo adjoint solver is used in four different settings. In each setting, the number of random walks starting from each grid point is respectively 1, 10, 100 and 1000. Griewank's checkpointing adjoint solver is used to compute the exact adjoint solution. Figure 2.1 plots the Monte Carlo adjoint equation in each of the four cases on top of the exact solution. The Monte Carlo adjoint solution clearly converges toward the exact solution as the number of random walks increases.

The rate of this convergence is depicted in Figure 2.2. The slope of the line in the log-log plot is roughly 0.5, indicating that our Monte Carlo adjoint solver converges at a rate of $\sqrt{q}$ (q is the number of random walk), which is the common rate of convergence in Monte Carlo methods.

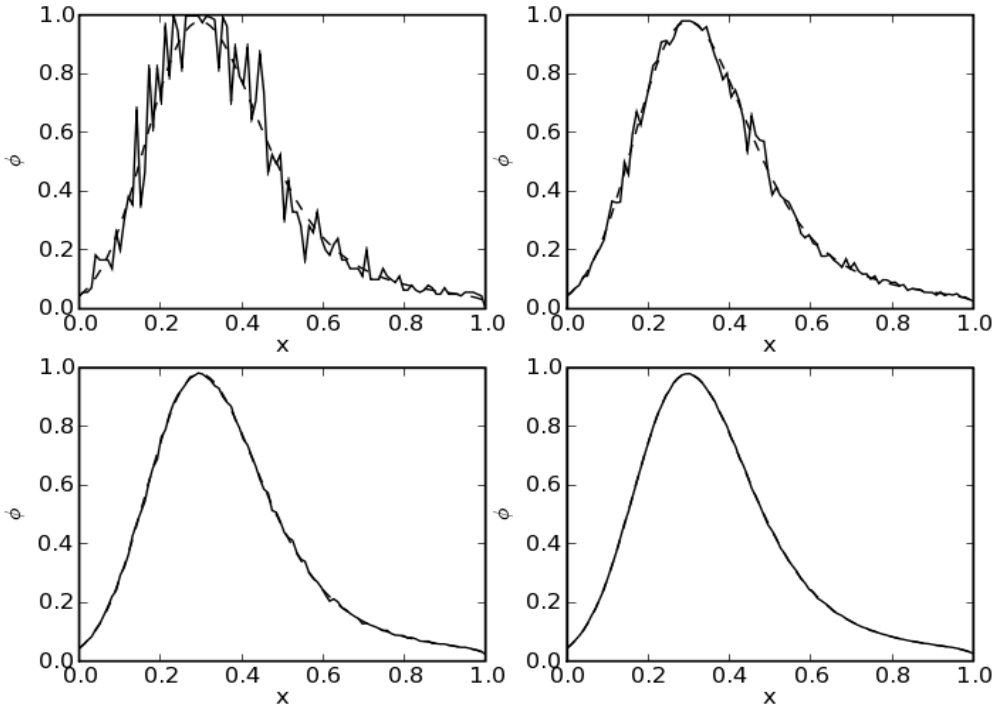

Figure 2.1: Solutions of the adjoint equation at time 0. Solid lines are solutions estimated by Monte Carlo method; dashed lines are the exact solution. The number of random walks starting from each grid point is: top-left plot: 1; top-right plot: 10; bottom-left plot: 100; bottom-right plot: 1000.

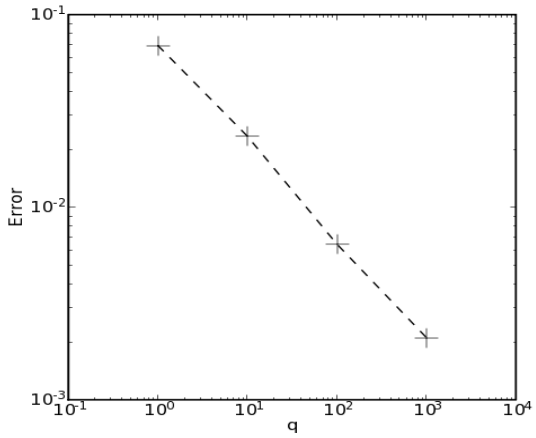

Figure 2.2: Convergence of Monte Carlo adjoint solution. The horizontal axis is q, the number of random walks starting from each grid point; the vertical axis is the L^2 distance between Monte Carlo adjoint solution and the exact solution.

2.7.2 Scaling efficiency

By avoiding trajectory storage and recomputation of the primal equation, our Monte Carlo adjoint solver is especially competitive to exact solution methods when the number of time steps is large. This efficiency is demonstrated by the following experiment.

In this experiment, the Burgers' equation is solved in the time interval $[0, 0.25]$ with uniform grids of 10 different sizes, ranging from $N = 10$ to 10,000. The CFL number for time integration is set to 0.5, thus the number of time steps increases proportionally to the grid size. Similar to the first experiment, the initial condition for Burgers' equation is (2.35); the numerical scheme used is first-order up-winding finite volume. Again, we solve the adjoint equation in the time interval $[0, 0.25]$ using both our Monte Carlo solver and Griewank's optimal checkpointing scheme. Two different settings, with $q = 1$ and $q = 10$, respectively (q is the number of random walks starting from each grid point), are used in the Monte Carlo solver. We then compare the time it takes using our Monte Carlo method in both settings to the time it takes using the optimal checkpointing scheme.

From the log-log plot in Figure 2.3, we observe that the computation time of our Monte Carlo method in both settings is proportional to the square of the grid size. The reason for this is that, with fixed CFL number, the number of time steps grows linearly with respect to the grid size, and the computational time is proportional to the product of the grid size and

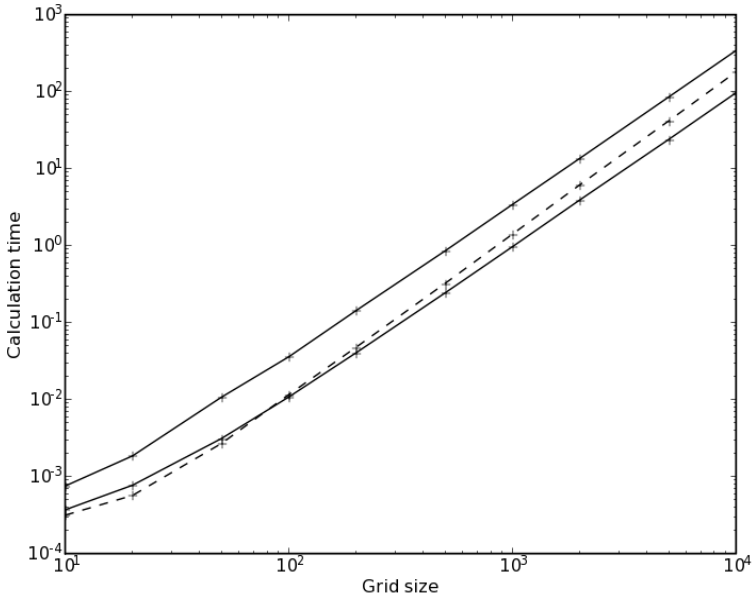

Figure 2.3: The computation time (in seconds) of the adjoint solution as grid size increases. The dashed line is the time it takes to compute the exact adjoint solution using Griewank's optimal checkpointing scheme; the upper solid line is the amount of time it takes to estimate the adjoint solution with the Monte Carlo method using $q = 10$ random walks per grid point; the lower solid line is the amount of time it takes to estimate the adjoint solution with the Monte Carlo method using $q = 1$ random walks per grid point. The calculation is done on a desktop computer with two Intel(R) Xeon(TM) 3.00GHz CPUs, 2GB memory, GNU/Linux 2.6.9-42.0.10.ELsmp. All calculations are single threaded.

the number of time steps. In contrast, the computation time of the optimal checkpointing scheme grows faster, with a theoretical rate of $n^2 \log n$, where n is the grid size.

This scaling efficiency of our Monte Carlo method is obtained without sacrificing the accuracy of its estimated solution. Figure 2.4 shows the L^2 distance between the Monte Carlo adjoint solution and the exact solution for different grid sizes. As the grid size and number of time steps increases, the quality of the Monte Carlo adjoint solution increases for a fixed number of random walks per grid point. This result indicates that when the computation upscales as the spatial and temporal resolution increases, our Monte Carlo method becomes more computationally efficient and produces increasingly accurate estimated adjoint solutions.

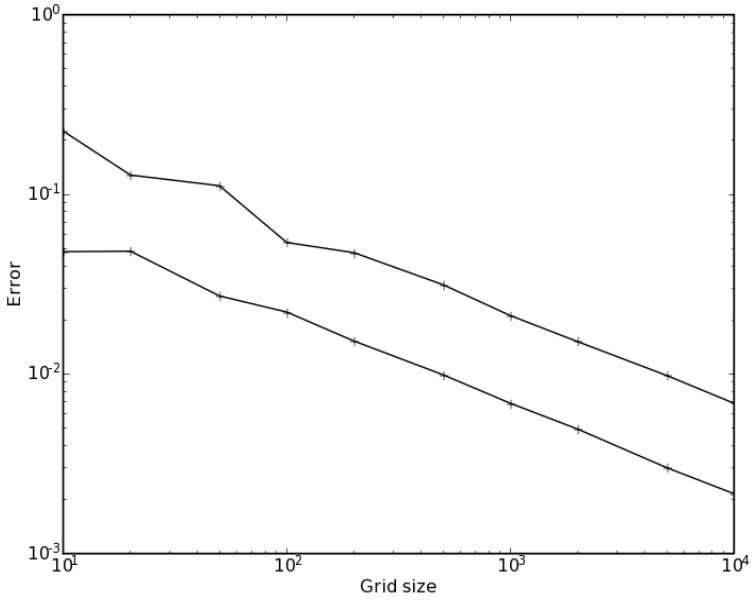

Figure 2.4: L^2 estimation error of the Monte Carlo adjoint solver. The top line is the error for $q = 1$ random walks per grid point; the bottom line is the error for $q = 10$ random walks per grid point.

2.7.3 A Monte Carlo adjoint-driven inverse problem

In this experiment, we test the performance of our Monte Carlo adjoint solver in solving a simple inverse problem: finding the initial condition of a Burgers' equation so that the solution at time $T = 0.25$ matches a prescribed target function

$$f_t(x) = \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right), \qquad x \in [0, 1],$$

where $\mu = 0.5$ and $\sigma = 0.2$. The boundary condition is Dirichlet at both $x = 0$ and $x = 1$.

We solve the Burgers' equation using first-order up-winding finite volume scheme on 100 uniformly spaced grid points; the CFL number is set to 0.5. The adjoint solution is obtained by our Monte Carlo adjoint solver; the number of random walks starting from each grid point is set to 1. This adjoint solution is used to drive a gradient-based optimization procedure to minimize the square L^2 difference between the solution at T and the prescribed

function f_t, thus solving the inverse problem. In this experiment, we use the BFGS quasi-Newton algorithm provided by Python module scipy.optimize. The initial guess fed into the optimization routine is f_t.

Due to insufficient precision of the gradient, calculated from the Monte Carlo adjoint solution, the optimization procedure terminated after 28 iterations without reaching its default error tolerance. The result of this optimization procedure and the corresponding solution at T is plotted in Figure 2.5. For the purpose of comparison, a fully converged solution at iteration 68 driven by the exact adjoint solution is plotted in figure 2.6. Despite the wiggling fluctuations on the solution using the Monte Carlo adjoint solver, the shape of the solution is captured. The corresponding Burgers' solution at T is also close to the target function. Moreover, the objective function, the square L^2 distance to the target function, is 2.4×10^{-4}, more than 2 orders of magnitude smaller than that of the initial guess, which is 7.5×10^{-2}. This result indicates that the Monte Carlo adjoint solution, being an approximation itself, is useful in obtaining approximate solutions of optimization and inverse problems. The accuracy of the adjoint solution limits the accuracy of the computed gradient, preventing full convergence of gradient-based optimization procedures.

Better convergence can be obtained if a smoothed version of the Monte Carlo adjoint solution is used to calculate the gradient. Figure 2.7 shows the solution of the same inverse problem driven by Monte Carlo adjoint solutions smoothed by a Gaussian filter. The number of random walks starting from each grid point q is still set to 1. The *sigma* of the Gaussian filter is 0.03. This time the optimization routine terminates after 58 iterations. Although the default error tolerance is still not reached, the quality of the solution is significantly improved. The final objective function is 3.2×10^{-5}, which is also an order of magnitude smaller than the final objective function of the non-smoothed case.

Apparently, despite being less accurate than the exact solution, the Monte Carlo adjoint is capable of reducing the objective function in optimization and inverse problems. In some practical problems where the Monte Carlo adjoint solver is more computationally efficient, it may be desirable to use the Monte Carlo adjoint solution to drive the optimization until further improvement is limited by its accuracy, then switch to an exact adjoint solver to ensure full convergence.

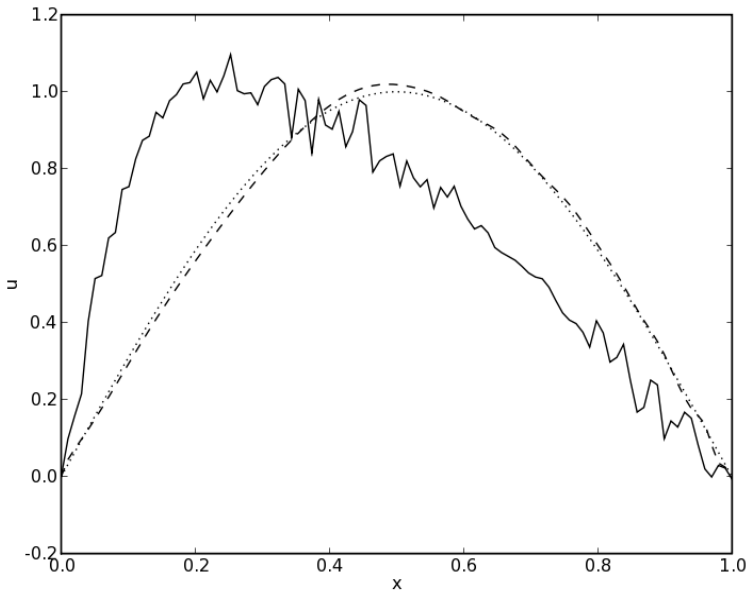

Figure 2.5: Solving an inverse problem using Monte Carlo adjoint solver. The solid line is the solution to the inverse problem after 28 BFGS iterations. The dashed line is the solution at time T of the Burgers' equation with the solid line as its initial condition. The dotted line is the target function f_t. The dashed line and dotted line being close means that the solid line is an approximate solution to the inverse problem.

2.8 Conclusion

The Monte Carlo method is an efficient method to approximate the solution of the adjoint equation. Its biggest advantages are:

1. By only advancing forward in time, it avoids storing the full trajectory or recomputing the original problem.

2. It is easy to compute only part of the adjoint solution, or a linear function of it. This saves computational time in practice.

3. It is conceptually easy to parallelize.

As we have demonstrated through our numerical experiments, our Monte Carlo method has better scaling efficiency than exact solution methods. By sacrificing some accuracy, choosing Monte Carlo adjoint solver in large-scale calculations can be advantageous in

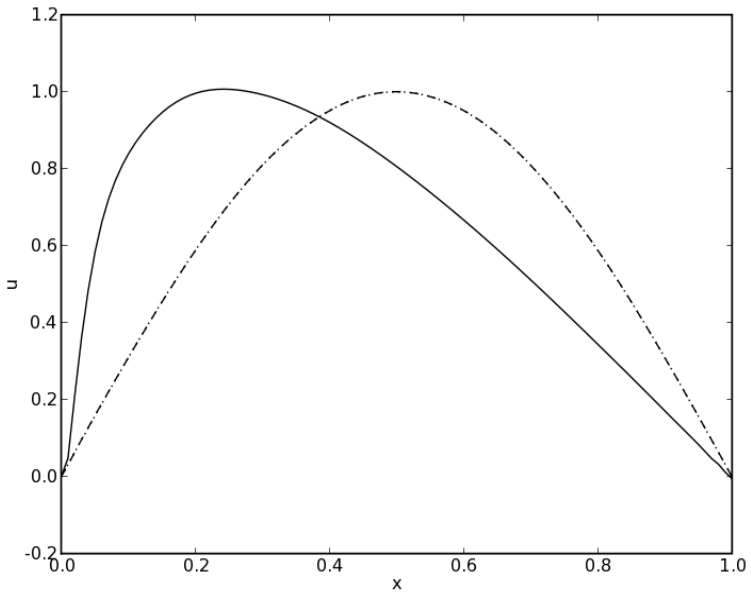

Figure 2.6: Solving the same inverse problem using exact adjoint solution. The solid line is the solution to the inverse problem after 68 BFGS iterations (fully converged). The dashed line is the solution at time T of the Burgers' equation with the solid line as its initial condition. The dotted line is the target function f_t. The dashed line and dotted line lying on top of each other means that the solid line is an accurate solution of the inverse problem.

terms of computation time and memory usage. This has been demonstrated in Burgers' equation.

Additional issues of this work remain to be resolved. The $O(\sqrt{n})$ convergence rate of the Monte Carlo method (2.21) makes it unattractive when the variance of the estimator D is high. This is the case when we apply our method to vector transport equations such as Navier Stokes equations. Therefore, future work should focus on reducing the variance of the estimator.

1. The variance of our estimator may be reduced by trying more advanced preconditioners (Tan, 2002) (Srinivasan, 2003), which can improve the convergence of the Neumann series. We are particularly interested in investigating preconditioners in the case of vector partial differential equations, such as Navier-Stokes equation.

2. We are interested in investigating Monte Carlo methods that are not based on the

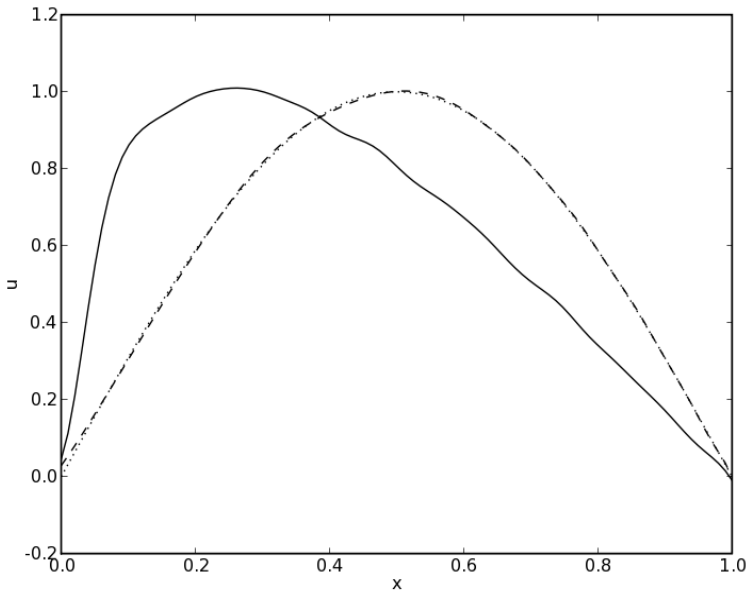

Figure 2.7: Solving an inverse problem using smoothed Monte Carlo adjoint solver. The solid line is the solution to the inverse problem after 57 BFGS iterations. The dashed line is the solution at time T of the Burgers' equation with the solid line as its initial condition. The dotted line is the target function f_t. The dashed line and dotted line being almost on top of each other means that the solid line is an accurate solution to the inverse problem.

Neumann series. These methods are especially useful in case the Neumann series have poor convergence.

Our ultimate goal is to use this method in large simulations of physical systems, such as aerodynamic simulations, turbulence simulations and fluid-structure coupled simulations. If this method can be efficiently generalized to vector transport equations, it will make very large scale calculations of their adjoint equations feasible.

Chapter 3

Dynamic Checkpointing Scheme

3.1 Introduction

The main challenge in solving the adjoint equations for time-dependent systems results from their time-reversal characteristic. Although a forward-time Monte Carlo algorithm has been proposed in Chapter 2, checkpointing schemes remain the dominant method to address this challenge. For a general nonlinear dynamical system (referred to here as the original system as opposed to the adjoint system)

$$\dot{u} = G(u,t), \quad u(0) = u_0, \quad 0 \leq t \leq T,$$

its adjoint equation is a linear dynamical system with a structure similar to the original system, except that the adjoint equation is initialized at time T. It then evolves backward in time, such that

$$\dot{q} = A(u,t)q + b(u,t), \quad q(T) = q_0(u(T)), \quad 0 \leq t \leq T.$$

While the specific forms of q_0, A and b depend on the predefined objective function, the required procedure to solve the adjoint equation is the same: to initialize the adjoint equation, $u(T)$ must first be obtained; as the time integration proceeds backward in time, the solution of the original system u is needed from $t = T$ backward to $t = 0$. At each time step of the adjoint time integration, the solution of the original system at that time step must be either already stored in memory or recalculated from the solution at the last stored time step. If we have sufficient memory to store the original system at all the time steps,

the adjoint equation can be solved without recalculating the original system. High-fidelity scientific and engineering simulations, however, require both many time steps and large memory to store the solution at each time step, making the storage of the solution at all the time steps impractical. Checkpointing schemes, in which only a small number of time steps are stored, apply naturally in this situation. Surprisingly, checkpointing does not necessarily take more computing time than storing the solution at all time steps, since the modified memory hierarchy may compensate for the required recalculations (Kowarz & Walther, 2006).

Checkpointing schemes can significantly reduce the memory requirement but increase computation time (Charpentier, 2001). In the first solution of the original system, the solution fields at a small set of carefully chosen time steps called checkpoints are stored. During the following adjoint calculation, the discarded intermediate solutions are then recalculated by solving the original system restarting from their last checkpoints. Old checkpoints are discarded once they are no longer useful, while new checkpoints are created in the recalculation process and replace the old ones. Griewank (Griewank, 1992) proposed a binomial checkpointing algorithm for certain values of the number of time steps. This recursive algorithm, known as **revolve**, has been proven (Griewank & Walther, 2000) to minimize the number of recalculations for any given number of allowed checkpoints and all possible number of time steps. With δ number of allowed checkpoints and τ recalculations, $\binom{\delta+\tau}{\tau}$ time steps can be integrated. The **revolve** algorithm achieves logarithmic growth of spatial and temporal complexity with respect to the number of time steps, acceptable for the majority of scientific and engineering simulations.

Griewank's **revolve** algorithm assumes *a priori* knowledge of the number of time steps, which represents an inconvenience, perhaps even a major obstacle in certain applications. In solving hyperbolic partial differential equations, for example, the size of each time step Δt can be constrained by the maximum wavespeed in the solution field. As a result, the number of time steps taken for a specified period of time is not known until the time integration is completed. A simple, but inefficient workaround is to solve the original system once for the sole purpose of determining the number of time steps, before applying the **revolve** algorithm. Two algorithms have been developed to generate checkpoints with unknown number of time steps *a priori*. The first is Hinze and Sternberg's **a-revolve** (Hinze & Sternberg, 2005). For a fixed number of allowed checkpoints, the **a-revolve** algorithm maintains a cost function that approximates the total computation time of recalculating

all intermediate solutions based on current checkpoint allocation. As time integration proceeds, the heuristic cost function is minimized by reallocating the checkpoints. Although *a priori* knowledge of the number of time steps is not required, the **a-revolve** algorithm is shown to be only slightly more costly than Griewank's **revolve** algorithm. However, it is difficult to prove a theoretical bound with this algorithm, making its superiority to the simple workaround questionable. The other algorithm is Heuveline and Walther's online checkpointing algorithm (Heuveline & Walther, 2006). This algorithm theoretically proves to be optimal; however, it explicitly assumes that the number of time steps is no more than $\binom{\delta+2}{\delta}$, and throws an error when this assumption is violated. This upper bound in the number of time steps is extended to $\binom{\delta+3}{\delta}$ by Stumm and Walther in their recent work (Stumm & Walther, 2008). This limitation makes their algorithm useless the number of time steps is large and the memory is limited.

We propose a dynamic checkpointing algorithm for solving adjoint equations. This algorithm, compared to previous ones, has three main advantages: First, unlike Griewank's **revolve**, it requires no *a priori* knowledge of the number of time steps, and is applicable to both static and adaptive time-stepping. Secondly, In contrast to Hinze and Sternberg's **a-revolve**, our algorithm is theoretically optimal in that it minimizes the repetition number τ, defined as the maximum number of times a specific time step is evaluated during the adjoint computation. As a result, the computational cost has a theoretical upper-bound. Thirdly, our algorithm works for an arbitrary number of time steps, unlike previous online checkpointing algorithms which limits the length of the time integration. When the number of time steps is less than $\binom{\delta+2}{\delta}$, our algorithm produces the same set of checkpoints as Heuveline and Walther's result; as the number of time steps exceeds this limit, our algorithm continues to update the checkpoints and ensures their optimality in the repetition number τ. We prove that for arbitrarily large number of time steps, the maximum number of recalculations for a specific time step in our algorithm is as small as possible. This new checkpointing algorithm combines the advantages of previous algorithms without their drawbacks.

As with **revolve**, our dynamic checkpointing algorithm is applicable both in solving adjoint equations and in automatic differentiation (AD). **revolve** and other static checkpointing algorithms can be inefficient in differentiating certain types of source code whose length of execution is *a priori* uncertain. Examples include "while" statements where the number of loops to be executed is only determined during execution, and procedures with

large chunks of code in "if" statements. These cases may seriously degrade the performance of static checkpointing AD schemes. Our dynamic checkpointing algorithm can solve this issue by achieving close to optimal efficiency regardless of the length of execution. Since most source codes to which automatic differentiation is applied are complex, including our algorithm can significantly increase the overall performance of automatic differentiation software.

The rest of this chapter is organized as follows. In Section 3.2 we first demonstrate our checkpoint generation algorithm. We prove the optimality of the algorithm, based on an assumption about the adjoint calculation procedure. Section 3.3 proves this assumption by introducing and analyzing our full algorithm for solving adjoint equations, including checkpoint generation and checkpoint based recalculations in backward adjoint calculation. The performance of the algorithm is theoretically analyzed and experimentally demonstrated in Section 3.4. Conclusions are provided in Section 3.5.

3.2 Dynamic checkpointing algorithm

The key concepts of **time step index**, **level** and **dispensability** of a checkpoint must be defined before introducing our algorithm. The time step index used to label the intermediate solutions of both the original equation and the adjoint equation at each time step. It is 0 for the initial condition of the original equation, 1 for the solution after the first time step, and increments as the original equation proceeds forward. For the adjoint solution the time step index decrements and reaches 0 for the last time step of the adjoint time integration. The checkpoint with time step index i, or equivalently, the checkpoint at time step i, is defined as the checkpoint that stores the intermediate solution with time step index i.

We define the concept of **level** and **dispensability** of checkpoints in our dynamic checkpointing algorithm. As each checkpoint is created, a fixed level is assigned to it. We will prove in Section 3.3 that this level indicates how many recalculations are needed to reconstruct all intermediate solutions between the checkpoint and the previous checkpoint of the same level or higher. We define a checkpoint as **dispensable** if its time step index is smaller than another checkpoint of higher level. When a checkpoint is created, it has the highest time step index, and is therefore not dispensable. As new checkpoints are added, existing checkpoints become dispensable if a new checkpoint has higher level. Our algorithm allocates and maintains δ checkpoints and one placeholder based on the level of

the checkpoints and whether they are dispensable. During each time step, a new checkpoint is allocated. If the number of checkpoints exceeds δ, an existing dispensable checkpoint is removed to maintain the specified number of checkpoints.

Algorithm 1 Dynamic allocation of checkpoints

Require: $\delta > 0$ given.
  Save time step 0 as a checkpoint of level ∞;
  for $i = 0, 1, \ldots$ **do**
    if the number of checkpoints $<= \delta$ **then**
      Save time step $i + 1$ as a checkpoint of level 0;
    else if At least one checkpoint is dispensable **then**
      Remove the dispensable checkpoint with the largest time step index;
      Save time step $i + 1$ as a checkpoint of level 0;
    else
      $l \Leftarrow$ the level of checkpoint at time step i;
      Remove the checkpoint at time step i;
      Save time step $i + 1$ as a checkpoint of level $l + 1$;
    end if
    Calculate time step $i + 1$ of the original system;
  end for

Two simplifications have been made in Algorithm 1. First of all, the algorithm maintains $\delta + 1$ checkpoints, one more than what is specified. This is because the last checkpoint is always at time step $i + 1$, where the solution has yet to be calculated. As a result, the last checkpoint stores no solution and takes little memory space (referred to here as a placeholder checkpoint), making the number of real checkpoints δ. The other simplification is the absence of the adjoint calculation procedures, which is addressed in Section 3.3.

In the remaining part of this section, we calculate the growth rate of the checkpoint levels as the time step i increases. This analysis is important because the maximum checkpoint level determines the maximum number of times of recalculation in the adjoint calculation τ, as proven in section 3.3. Through this analysis, we show that our algorithm achieves the same number of time steps as Griewank's **revolve** for any given δ and τ. Since **revolve** is known to maximize the number of time steps for fixed number of checkpoints δ and times of calculations τ, our algorithm is thus optimal in the same sense.

Algorithm 1 starts by saving time step 0 as a level infinity checkpoint, and time steps 1 through time step δ as level 0 checkpoints. When $i = \delta$, there are already $\delta + 1$ checkpoints, none of which are dispensable. The algorithm enters the **else** clause with $l = 0$. In this

clause, the checkpoint at time step δ is removed, and time step $\delta + 1$ is saved as a level 1 checkpoint, making all checkpoints except for the first and last dispensable. As the time integration continues, these $\delta - 1$ dispensable checkpoints are then recycled, while time steps $\delta + 2$ to 2δ take their place as level 0 checkpoints. A third checkpoint of level 1 is created for time step $2\delta + 1$, while the remaining $\delta - 2$ level 0 checkpoints become dispensable. Each time a level 1 checkpoint is made, one less level 0 checkpoint is dispensable, resulting in one less space between the current and the next level 1 checkpoints. The $\delta + 1$st level 1 checkpoint is created for time step $(\delta+1)+\delta+(\delta-1)+\ldots+2 = \binom{\delta+2}{2} - 1$. At this point, all $\delta + 1$ checkpoints are level 1, while the space between them is an arithmetic sequence. All checkpoints created thus far are exactly the same as Heuveline and Walther's (Heuveline & Walther, 2006) online checkpointing algorithm, although their algorithm breaks down and produces an error at the very next time step.

Our algorithm continues by allocating a level 2 checkpoint for time step $\binom{\delta+2}{2}$. At the same time, the level 1 checkpoint at time step $\binom{\delta+2}{2} - 1$ is deleted, and all other level 1 checkpoints become dispensable. A similar process of creating level 2 checkpoints ensues. The third level 2 checkpoint is created for time step $\binom{\delta+2}{2} + \binom{\delta+1}{2}$, after the same evolution described in the previous paragraph with only $\delta - 1$ free checkpoints. The creation of level 2 checkpoints continues until the $\delta + 1$st level 2 checkpoint is created with time step $\binom{\delta+2}{2} + \binom{\delta+1}{2} + \ldots + \binom{3}{2} = \binom{\delta+3}{3} - 1$. The creation of the first level 3 checkpoint follows at time step $\binom{\delta+3}{3}$. Figure 3.1 illustrates an example of this process where $\delta = 3$. Up to now, we found that the time steps of the first level 0, 1, 2 and 3 checkpoints are respectively, 1, $\delta + 1$, $\binom{\delta+2}{2}$, and $\binom{\delta+3}{3}$. This interesting pattern leads us to our first proposition.

Proposition 3.2.1. *In Algorithm 1, the first checkpoint of level τ is always created for time step $\binom{\delta+\tau}{\tau}$.*

To prove this proposition, we note that it is a special case of the following lemma when $i = 0$, making it only necessary to prove the lemma.

Lemma 3.2.2. *In Algorithm 1, let i be the time step of a level $\tau - 1$ or higher checkpoint. The next checkpoint with level τ or higher is at time step $i + \binom{\delta - n_i + \tau}{\tau}$ and is level τ, where n_i is the number of indispensable checkpoints allocated before time step i.*

Proof. We use induction here. When $\tau = 0$, $\binom{\delta - n_i + \tau}{\tau} = 1$. The next checkpoint is allocated at time step $i + 1$, and its level is non-negative; therefore, it is level 0 or higher. If there is a dispensable checkpoint at time step i, the new checkpoint is level 0; otherwise, it is level 1.

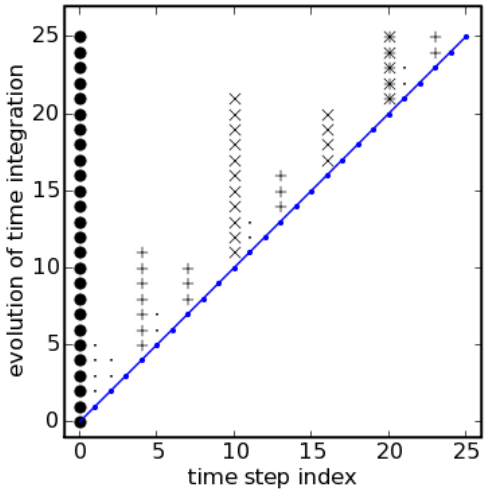

Figure 3.1: Dynamic allocation of checkpoints for $\delta = 3$. The plot shows the checkpoints' distribution during 25 time steps of time integration. Each horizontal cross-section on the plot represents a snapshot of the time integration history, from time step 0 to time step 25, indicated by the y-axis. Different symbols represent different levels of checkpoints: Circles are level ∞ checkpoint at time step 0. Thin dots, "+", "X" and star symbols corresponds to level 0, 1, 2 and 3 checkpoints, respectively. The thick dots connected by a line indicate the time step index of the current solution, which is also the position of the placeholder checkpoint.

Assuming that the lemma holds true for any $0 \leq \tau < \tau_0$, we now prove it for $\tau = \tau_0$. First, if $n_i = \delta$, $\binom{\delta - n_i + \tau}{\tau} = 1$. No dispensable checkpoint exists in this case; thus, the **else** clause is executed to create a higher level checkpoint at time step $i+1$. On the other hand, if $n_i < \delta$, we use the induction hypothesis. The next level $\tau - 1$ checkpoint is created at time step

$$i_1 = i + \binom{\tau - n_i + \tau - 1}{\tau - 1},$$

incrementing the number of indispensable checkpoints

$$n_{i_1} = n_i + 1.$$

As a result, the following level $\tau - 1$ checkpoints are created at time steps

$$i_2 = i_1 + \binom{\delta - n_{i_1} + \tau - 1}{\tau - 1},$$

$$i_3 = i_2 + \binom{\delta - n_{i_2} + \tau - 1}{\tau - 1},$$
$$\dots$$
$$i_{k+1} = i_k + \binom{\delta - n_{i_k} + \tau - 1}{\tau - 1},$$
$$\dots$$

This creation of level $\tau-1$ checkpoints continues until $n_{i_k} = n_i + k = \delta + 1$. At this point, all existing checkpoints are level $\tau - 1$. Consequently, the **else** clause is executed with $l = \tau - 1$ and creates the first level τ checkpoint at time step

$$i_{\delta - n_i + 1} = i + \binom{\delta - n_i + \tau - 1}{\tau - 1} + \binom{\delta - n_i - 1 + \tau - 1}{\tau - 1} + \ldots + \binom{\tau - 1}{\tau - 1}$$

Plug in Pascal's rule

$$\binom{m}{k} = \binom{m-1}{k} + \binom{m-1}{k-1} = \sum_{i=0}^{k} \binom{m-1-i}{k-i}$$

with $m = \delta - n_i + \tau$ and $k = \delta - n_i$, this equation simplifies to

$$i_{\delta - n_i + 1} = i + \binom{\delta - n_i + \tau}{\tau},$$

which completes the induction. □

For those who are familiar with Griewank's **revolve** algorithm, note that the checkpoints produced by our algorithm and the **revolve** algorithm are identical when the number of time steps equals to $\binom{\tau+\delta}{\tau} - 1$ for any τ. In this case, our algorithm generates δ level τ checkpoints, whose time step indices are separated with distances $\binom{\delta+\tau-i}{\delta}, i = 0, 1, \ldots, \delta-1$, respectively. On the other hand, the checkpoints created by the **revolve** algorithm have the same distance of separation. Although this looks like a coincident, it actually indicates the optimality of both algorithms in this case. [1]

Theorem 3.2.1 leads us to the key conclusion of our theoretical analysis.

Corollary 3.2.3. *For any $\tau > 0$ and $\delta > 0$, η time steps can be taken in Algorithm 1*

[1] Through some analysis, one can show that this checkpoint pattern is the only optimal one for this number of time steps.

without any checkpoint being raised to level $\tau + 1$*, if and only if*

$$\eta \leq \binom{\delta + \tau + 1}{\tau + 1}$$

Combined with Theorem 6.1 of Griewank's paper (Griewank, 1992), Corollary 3.2.3 indicates that the checkpoints produced by algorithm 1 has the optimal repetition number, determined by

$$\binom{\delta + \tau}{\tau} < \eta \leq \binom{\delta + \tau + 1}{\tau + 1}.$$

This conclusion is based on the fact that

$$\textbf{maximum checkpoint level} \geq \textbf{repetition number}. \tag{3.1}$$

The repetition number here does not count the first time the original system is solved. The next section will prove this equality by analyzing the adjoint calculation algorithm.

3.3 Adjoint calculation

In this section, we fully describe our algorithm of solving adjoint equations using dynamic checkpointing. The algorithm consists of a forward sweep and a backward sweep. The forward sweep solves the original system and stores intermediate solutions at checkpoints. The adjoint system is then solved in the backward sweep, using the stored checkpoints to initialize recalculations. Algorithm 2 describes this high-level scheme; the details of checkpoint manipulations and recalculations are given in Algorithms 3 and 4.

The algorithm for advancing the original system is essentially identical to Algorithm 1. In the first forward sweep, checkpoints generated by repeatedly calling Algorithm 3 satisfy Lemma 3.2.2, Theorem 3.2.1, and hence Corollary 3.2.3. Moreover, a recalculation sweep consisting of a series of calls to Algorithm 3 also satisfies Lemma 3.2.2. In each time step, the solution at time step i, instead of time step $i + 1$, is stored. This strategy ensures that the last checkpoint at time step $i + 1$ is always a placeholder checkpoint. Although our algorithm updates and maintains $\delta + 1$ checkpoints, only δ of them store solutions.

Compared to checkpoint allocation, retrograding the adjoint system is relatively simple. Solving the adjoint solution at time step i requires both the solution to the adjoint system at time step $i + 1$ and the solution to the original system at time step i. The latter can be

Algorithm 2 High level scheme to solve the adjoint equation

Initialize the original system;
$i \Leftarrow 0$;
Save time step 0 as a placeholder checkpoint of level ∞;
while the termination criteria of the original system is not met **do**
  Solve the original system from time step i to $i+1$ using Algorithm 3;
  $i \Leftarrow i+1$;
end while
Initialize the adjoint system;
while $i >= 0$ **do**
  $i \Leftarrow i-1$;
  Solve the adjoint system from time step $i+1$ to i using Algorithm 4;
end while

Algorithm 3 Solving the original system from time step i to $i+1$

Require: $\delta > 0$ given; solution at time step i has been calculated.
if the number of checkpoints $<= \delta$ **then**
  Save time step $i+1$ as a checkpoint of level 0;
else if At least one checkpoint is dispensable **then**
  Remove the dispensable checkpoint with the largest time step index;
  Save time step $i+1$ as a checkpoint of level 0;
else
  $l \Leftarrow$ the level of checkpoint at time step i;
  Remove the checkpoint at time step i;
  Save time step $i+1$ as a checkpoint of level $l+1$;
end if
if time step i is in the current set of checkpoints **then**
  Store the solution at time step i to the checkpoint;
end if
Calculate time step $i+1$ of the original system.

directly retrieved if there is a checkpoint for time step i; otherwise, it must be recalculated from the last saved checkpoint. Note that this algorithm calls Algorithm 3 to recalculate

Algorithm 4 Solving the adjoint system from time step $i+1$ to i

Require: $\delta > 0$ given; adjoint solution at time step $i+1$ has been calculated.
  Remove the placeholder checkpoint at time step $i+1$;
  if the last checkpoints is at time step i **then**
    Retrieve the solution at time step i, making it a placeholder checkpoint;
  else
    Retrieve the solution at the last checkpoint, making it a placeholder checkpoint;
    Initialize the original system with the retrieved solution;
    Solve the original system to time step i by calling Algorithm 3;
  end if
  Calculate time step i of the adjoint system;

the solution of the original system at time step i from the last saved checkpoint, during which more checkpoints are created between the last saved checkpoint and time step i. These new checkpoints reduce the number of recalculations using memory space freed by removing checkpoints after time step i.

Figure 3.2 and 3.3 shows examples of the entire process of Algorithm 2 with four different values of δ. As can be seen, the less the specified number of checkpoints δ is, the more recalculations of the original equation are performed, and the longer it takes to solve the adjoint equation. When $\delta \geq 25$, there is enough memory to store every time step, so no recalculation is done. The maximum finite checkpoint level is 0 in this case. When $\delta = 6$, the maximum finite checkpoint level becomes 1, and at most 1 recalculation is done for each time step. When δ decreases to 5 and 3, the maximum finite checkpoint level increases to 2 and 3, respectively, and the maximum number of recalculations also increases to 2 and 3, respectively. From these examples, we see that the number of recalculations at each time step is bounded by the level of the checkpoints after that time step. In the remaining part of this section, we focus on proving this fact, starting with Lemma 3.3.1.

Lemma 3.3.1. *Denote τ_i as the highest level of all checkpoints whose time steps are greater than i. For any i, τ_i does not increase for each adjoint step. Furthermore, if i is between the time steps of the last two indispensable checkpoints before an adjoint step, τ_i decreases after this adjoint step.*

Proof. Fix i, consider an adjoint step from time step j to $j-1$, where $j > i$. Note that before this adjoint step, time step j is a placeholder checkpoint. Denote the level of this checkpoint

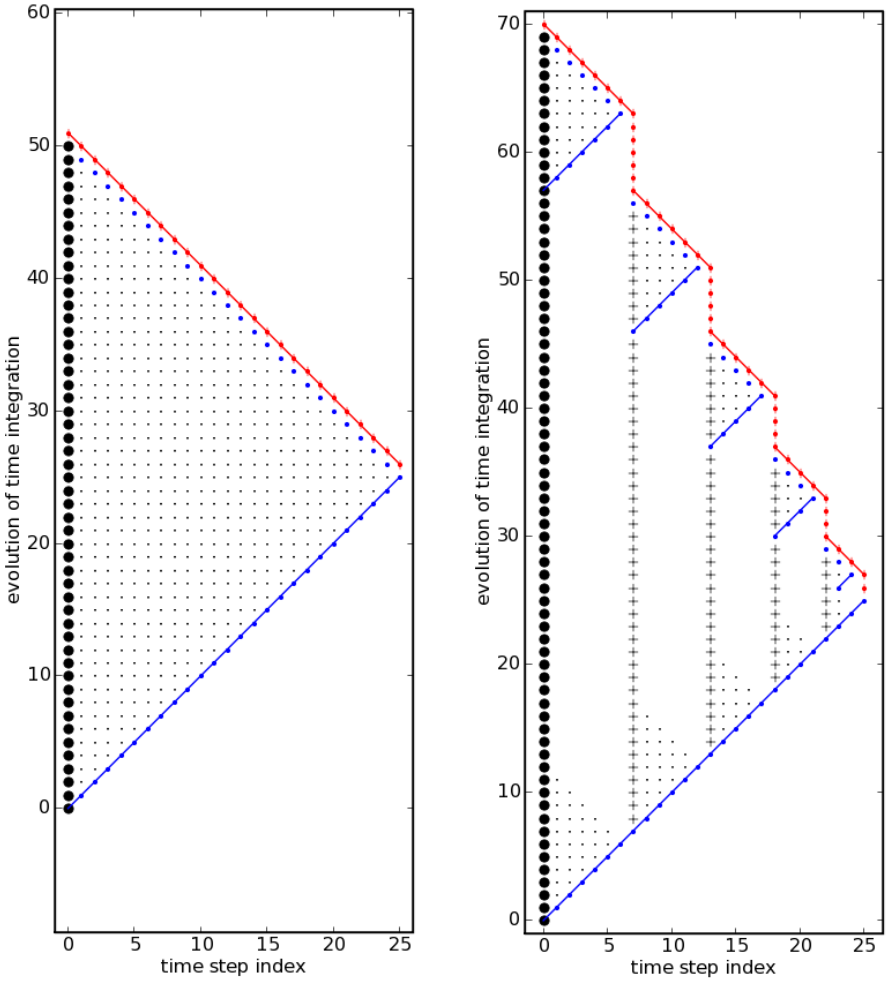

Figure 3.2: Distribution of checkpoints during the process of Algorithm 2. The left plot is for $\delta \geq 25$, the right plot is for $\delta = 6$. Each horizontal cross-section on the plot represents a snapshot of the time integration history, from the beginning of the forward sweep to the end of the adjoint sweep, indicated by the y-axis. Different symbols represent different levels of checkpoints: Circles are level ∞ checkpoint at time step 0. Thin dots, "+", "X" and star symbols correspond to level 0, 1, 2 and 3 checkpoints, respectively. The round thick dots indicate the time step index of the current original solution, which is also the position of the placeholder checkpoint; the lines connecting these round dots indicate where and when the original equation is solved. The thick dots with a small vertical line indicate the time step index of the current adjoint solution, while the lines connecting them indicate where and when the adjoint equation is solved.

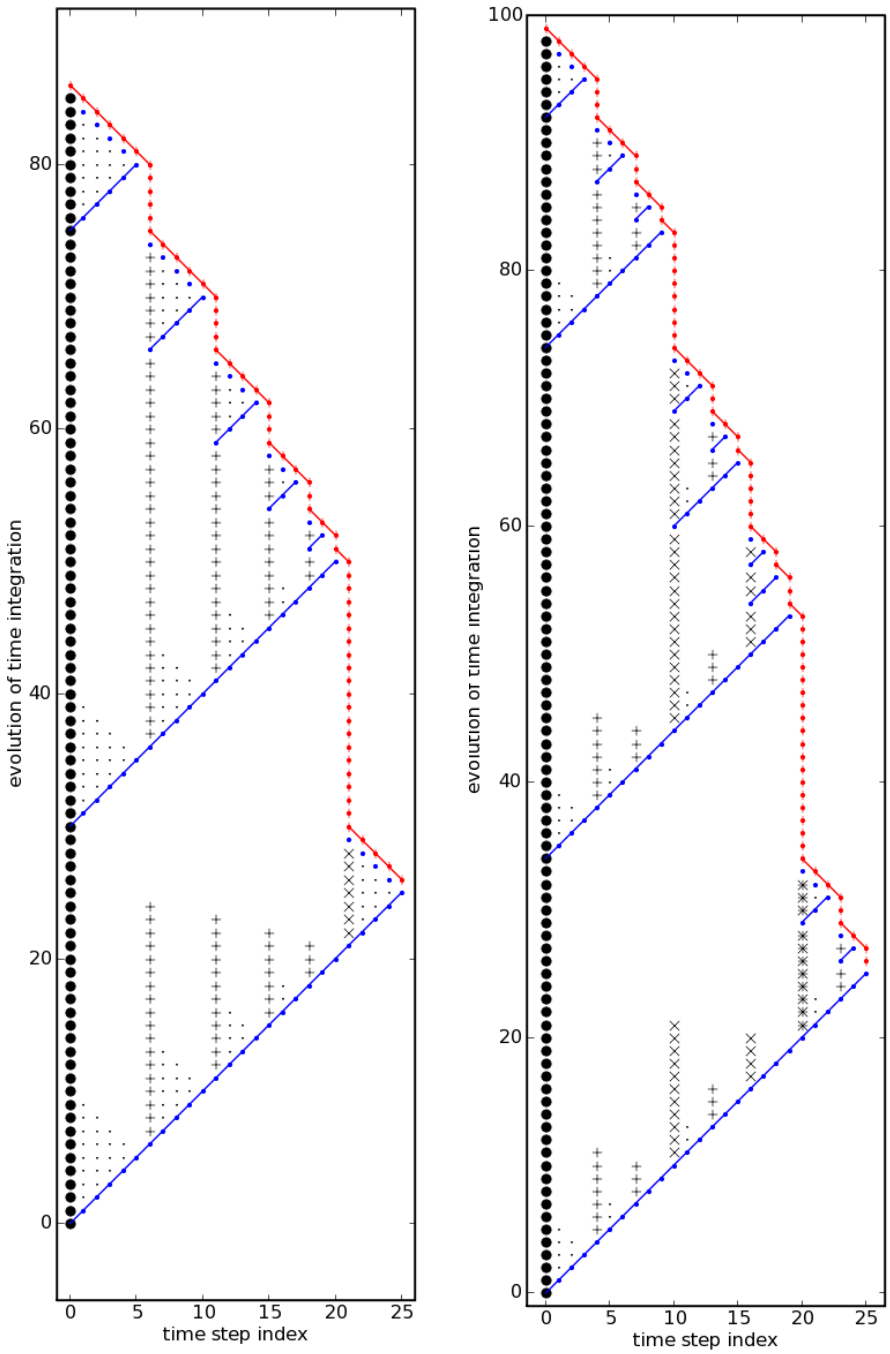

Figure 3.3: Distribution of checkpoints during Algorithm 2. $\delta = 5$ and 3 for left and right plot, respectively.

as l_j. If time step $j-1$ is stored in a checkpoint before this adjoint step, then checkpoint j is removed and all other checkpoints remain, making Lemma 3.3.1 trivially true. If time step $j-1$ is not stored before this adjoint step, the original system is recalculated from the last checkpoint to time step $j-1$. We now prove that this recalculation sweep does not produce any checkpoints with level higher or equal to l_j. As a result, τ_i does not increase; furthermore, if no other level τ_i checkpoint has time step greater than i, τ_i decreases.

Denote the time step of the last checkpoint as k, and its level as l_k. We use capital letter B to denote the recalculation sweep in the current adjoint step. Note that sweep B starts from time step k and ends at time step $j-1$. On the other hand, the checkpoint at time step j is either created during the first forward sweep, or during a recalculation sweep in a previous adjoint step. We use capital letter A to denote a part of this sweep from time step k to when the checkpoint at time step j is created. To compare the sweeps A and B, note the following two facts: First, the checkpoint at time step k exists before sweep A. This is because sweep A created the checkpoint at time step j, making all subsequent recalculations before sweep B start from either this checkpoint or a checkpoint whose time step is greater than j. Consequently, the checkpoint at time step k is not created after sweep A, it is created either by sweep A or exists before sweep A. Secondly, because any sweep between A and B starts at time step j or greater, it does not create any checkpoint with time step index smaller than k. On the other hand, any checkpoint removed during or after sweep A and before B is always the lowest level at that time, and thus, does not cause the increase of the number of dispensable checkpoints. As a result, the number of dispensable checkpoints with time step index smaller than k is no less at the beginning of sweep A than at the beginning of sweep B. This is identical to stating

$$n_k^A \geq n_k^B,$$

where n_k is the number of indispensable checkpoints with time step index less than k; the superscript identifies the sweep in which the indispensable checkpoints are counted.

Now, we complete the proof by comparing sweep A and B in following two cases: If $l_k < l_j$, we assert that sweep A does not create any higher level checkpoint than l_k at time steps between k and j. Suppose the contrary is true, that sweep A has created an at least level l_k+1 checkpoint between the two time steps. Because no checkpoint is between time steps k and j when sweep B starts, the supposed checkpoint must have been removed

before sweep B happens. But in that case, the checkpoint at time step k must be also removed, because its level is lower. This cannot happen because sweep B starts at this checkpoint. This contradiction proves our assertion. Because time step j is the first higher level checkpoint than l_k created by sweep A with larger time step index than k, its time step index, based on Lemma 3.2.2, is

$$j = k + \binom{\delta - n_k^A + l_k + 1}{l_k + 1}.$$

Since $n_k^A \geq n_k^B$, we further get

$$j \leq k + \binom{\delta - n_k^B + l_k + 1}{l_k + 1}.$$

Using Lemma 3.2.2 again, we conclude that the first checkpoint with level higher than l_k is at time step index greater or equal to j. But time step $j-1$ is the last step of sweep B; therefore, sweep B do not create any checkpoint with level higher than l_k. Since $l_k < l_j$, no level l_j or higher checkpoint is created by sweep B. This completes the proof in the first case.

In the second case, $l_k \geq l_j$, we assert that the checkpoint at time step j is the first one created by sweep A with level higher or equal to l_j. Suppose the contrary is true, that sweep A has created an at least level l_j checkpoint between the two time steps. Because no checkpoint is between time steps k and j when sweep B starts, the supposed checkpoint must have been removed before sweep B happens. But in that case, the checkpoint at time step j must be also removed, since it has the same level and is at a larger time step index. This is contradictory because sweep B starts at this checkpoint. Therefore, our assertion is true, and we get from Lemma 3.2.2

$$j \leq k + \binom{\delta - n_k^A + l_j}{l_j}.$$

Since $n_k^A \geq n_k^B$, we further get

$$j \leq k + \binom{\delta - n_k^B + l_j}{l_j}.$$

Using Lemma 3.2.2 again, we conclude that the first checkpoint with level higher or equal

to l_j is at time step index greater or equal to j. But time step $j-1$ is the last step of sweep B; therefore, sweep B does not create any checkpoint with level higher or equal to l_j. This completes the proof in the second case. □

Lemma 3.3.1 indicates that the highest level of all checkpoints whose time step index is greater than a particular value is monotone decreasing during the adjoint time integration. Equipped with this result, we prove the main proposition in this section.

Proposition 3.3.2. *Let $i \geq 0$. Time integrating the adjoint system from any time step $i' > i$ to time step i requires calling Algorithm 3 at most τ_i times for advancing from time step i to time step $i+1$, where τ_i is the highest level of all the checkpoints with time step indices greater than i before the adjoint time integration.*

Proof. To prove this proposition, we use induction again. If $\tau_i = 0$, then the **else** clause in Algorithm 3 is never executed after time step i. All checkpoints created after time step are i neither dispensable nor removed, including the $i+1$st time step. No recalculation from time step i to $i+1$ is necessary to obtain the solution at time step $i+1$, in which case the proposition holds.

Assuming that the proposition holds true for all $\tau_i \leq \mathcal{T}$, we prove it for $\tau_i = \mathcal{T}+1$. Among all level τ_i checkpoints with time step index larger than i, let j the smallest time step index of such checkpoints. Because at the beginning of the adjoint time integration, no checkpoint with time step index greater than i has higher level than τ_i, by Lemma 3.3.1, the entire adjoint time integration from time step i' to i does not produce any checkpoint with higher level than τ_i and time step index greater than i. As a result, the level τ_i checkpoint at time step j is not removed by any recalculation sweep until the adjoint time integration reaches time step j. Any recalculation sweep before the adjoint time integration reaches time step j starts either at time step j or from a checkpoints with greater time step than j. Therefore, recalculation at time step i is only possible after the adjoint time integration reaches time step j. We focus on this part of the adjoint calculation during the remainder of this proof.

If $i = k-1$, because the solution at $i+1 = k$ is no longer needed, no recalculation from i to $i+1$ is done. Otherwise, $j-1$ is not a checkpoint, and a recalculation sweep is performed from the last checkpoint to time step $j-1$. This sweep includes zero or one recalculation from time step i to time step $i+1$, depending on where it is initialized. Because k is the smallest time step index of level τ_i checkpoints with time step greater than i, time step i is

between the last two indispensable checkpoints before the adjoint step from time step j to $j-1$. This enables us to use the "furthermore" part of Lemma 3.3.1 and conclude that τ_i decreases to $\mathcal{T}$ or lower after this adjoint step. Therefore, from our induction hypothesis, the number of recalculations at time step i after this adjoint step at most $\tau_i - 1$. Combining this number with the possible 1 recalculation during the adjoint step from j to $j-1$, the total recalculations from time step i to time step $i+1$ during all the adjoint steps from time step i' to i is at most τ_i, completing our induction. □

As a special case of the proposition, consider time step i' to be the last time step in solving the original system, and $i = 0$. Let $\tau = \tau_0$ be the highest finite level of all checkpoints when the first forward sweep is completed. According to the proposition, $\tau_i \leq \tau$ for all i, resulting in the following corollary.

Corollary 3.3.3. *In Algorithm 3, let τ be the maximum finite checkpoint level after the original system is solved for the first time. Algorithm 4 is called at most τ times for each time step during the backward sweep in solving the adjoint equation.*

Corollary 3.3.3 effectively states Equation (3.1). Combined with Corollary 3.2.3, it proves that our dynamic checkpointing algorithm achieves a repetition number of τ for $\binom{\delta+\tau}{\tau}$ time steps. This, as proved by Griewank (Griewank, 1992), is the optimal performance for any checkpointing algorithm.

3.4 Algorithm efficiency

The previous two sections presented our dynamic checkpointing algorithm and its optimality in the repetition number. Here, we discuss the implication of this optimality on the performance of this algorithm, and demonstrate its efficiency using numerical experiments. We begin by providing a theoretical upper bound on the total number of time step recalculations in our adjoint.

Proposition 3.4.1. *The overall number of forward time step recalculations in adjoint calculation n_r is bounded by*

$$n_r < \tau\eta - \binom{\delta+\tau}{\tau-1}, \tag{3.2}$$

where η is the number of time steps, τ is the repetition number determined by

$$\binom{\delta+\tau}{\tau} < \eta \leq \binom{\delta+\tau+1}{\tau+1},$$

and δ is the number of allowed checkpoints.

Proof. Since the repetition number is τ, there is at least one checkpoint of level τ, and no checkpoint of higher level (Corollary 3.2.3). The first level τ checkpoint is created at time step index $\binom{\delta+\tau}{\tau}$, according to proposition 3.2.1. Since no checkpoint of higher level is created, this first level τ checkpoint can not removed by algorithm 1. We split the adjoint calculation at the first level τ checkpoint at time step index $\binom{\delta+\tau}{\tau}$. First, in calculating the adjoint steps of index from η to $\binom{\delta+\tau}{\tau}$, every forward step is recalculated at most τ times by definition of the repetition number τ. The total number of forward time step recalculations in this part is less or equal to

$$\tau\left(\eta - \binom{\delta+\tau}{\tau}\right).$$

In fact it is always less than, since the very last time step is never recalculated. Second, in calculating the adjoint steps of index from $\binom{\delta+\tau}{\tau} - 1$ to 0, if no checkpoint exists in this part, the total number of forward time step recalculations is (Griewank & Walther, 2000)

$$\tau\binom{\delta+\tau}{\tau} - \binom{\delta+\tau}{\tau-1}.$$

The total number of recalculations in this part is less than this number if there is one or more checkpoints between time step 0 and $\binom{\delta+\tau}{\tau}$. Therefore, the total number of forward time step recalculations in the entire adjoint calculation, which is the sum of the number of recalculations in the two parts, is less than

$$\tau\eta - \binom{\delta+\tau}{\tau-1}.$$

□

Having this upper bound, we can the total number of recalculations of our algorithm with the optimal static checkpointing scheme. The minimum number of total forward time

step calculations for an adjoint calculation of length η is (Griewank & Walther, 2000)

$$(\tau + 1)\,\eta - \binom{\delta + \tau + 1}{\tau},$$

including the $\eta - 1$ forward time step calculations before the adjoint calculation starts. Therefore, the minimum number of total recalculations is

$$n_r \geq n_{r\,opt} = \tau\eta - \binom{\delta + \tau + 1}{\tau} + 1. \tag{3.3}$$

Equations (3.2) and (3.3) bound the total number of forward time step recalculations of our dynamic checkpointing scheme. They also bound the deviation from optimality in terms of total recalculations.

$$n_r - n_{r\,opt} < \binom{\delta + \tau + 1}{\tau} - \binom{\delta + \tau}{\tau - 1} - 1 = \binom{\delta + \tau}{\tau} - 1$$

This bound naturally leads to the following corollary:

Corollary 3.4.2. *Using our dynamic checkpointing scheme takes less total recalculations than running the simulation forward, determine the number of time steps, and then (knowing the time step count) use the proven optimal* **revolve** *algorithm.*

Proof. Running the simulation forward to determine the number of time steps, then use **revolve** takes a total number of $n'_r = \eta + n_{r\,opt}$ recalculations. Since $\eta > \binom{\delta+\tau}{\tau}$, we have $n'_r > n_{r\,opt} + \binom{\delta+\tau}{\tau} > n_r$. □

With these theoretical bounds, we turn to study experimentally the actual number of recalculations of our dynamic checkpointing algorithm. Figures 3.4 plots the actual number of forward time step recalculations together with the upper and lower bound defined by Equations (3.2) and (3.3). The total number of recalculations is divided by the number of time steps, and the resulting average number of recalculations for each time step is plotted. As can be seen, the actual number lies between the lower and upper bound, as predicted by theory. Moreover, more points tend to be closer to the lower bound than to the upper bound, and some points lie exactly on the lower bound. This means that our algorithm in most cases outperforms what Corollary 3.4.2 guarantees.

While the total number of recalculations of our dynamic checkpointing algorithm is not necessarily as small as static checkpointing schemes, the repetition number is proven to be

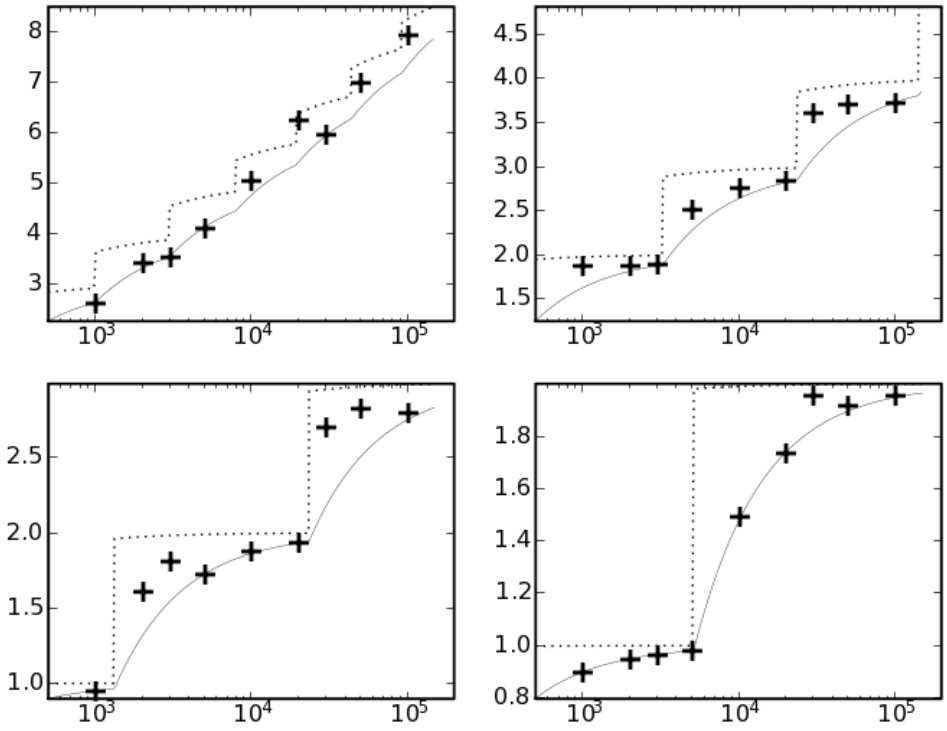

Figure 3.4: The x-axis is the number of time steps, and the y-axis is the average number of recalculations for each time step, defined as the total number of recalculations divided by the number of time steps. Plus signs are the actual average number of recalculations of the dynamic checkpointing scheme; the solid line and the dot line are the lower and upper bound defined by Equations (3.2) and (3.3). The upper-left, upper-right, lower-left and lower-right plots correspond to 10, 25, 50 and 100 allowed checkpoints, respectively.

optimal in any situation. Table 3.4 shows the maximum number of time steps our algorithm can proceed for a given number of checkpoints and number of recalculations (repetition number). The range of checkpoints are typical for most of today's unsteady simulations in fluid mechanics, which range from 10 checkpoints in high-fidelity multi-physics simulations where limited memory is a serious issue, to 100 checkpoints in calculations where memory requirements are less stringent. The maximum number of time steps is calculated by the formula $\binom{\delta+\tau}{\tau}$, where δ is the number of checkpoints, and τ the number of recalculations plus 1 (to include the first calculation of the original system). As can be seen, the number of time

steps grows very rapidly as either the checkpoints or the recalculations increase. With only 10 checkpoints, 5 to 7 recalculations should be sufficient for the majority of today's unsteady flow simulations. With 100 checkpoints, only one or two recalculations are needed to achieve the same number of time steps. With this many checkpoints, our algorithm requires only three recalculations for 4.6×10^6 time steps, much more than current calculations use.

Table 3.1: Maximum number of time steps for fixed number of checkpoints and repetition number τ.

	$\tau = 1$	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 10$
10 checkpoints	66	286	1001	3003	1.85×10^5
25 checkpoints	351	3276	23751	1.43×10^5	1.84×10^8
50 checkpoints	1326	23426	3.16×10^5	3.48×10^6	7.54×10^{10}
100 checkpoints	5151	1.77×10^5	4.60×10^6	9.66×10^7	4.69×10^{13}

To end this section, we use a numerical experiment to demonstrate that our algorithm is not only theoretically advantageous, but also highly efficient in practice. In this experiment, the original system is the inviscid Burgers' equation,

$$u_t + \frac{1}{2}\left(u^2\right)_x = 0, \quad x \in [0,1], \quad t \in [0,1];$$
$$u|_{t=0} = \sin 2\pi x, \quad u|_{x=0,1} = 0.$$

We discretize this partial differential equation with a first-order up-winding finite volume scheme with 250 mesh volumes, and use forward Euler time integration with a fixed CFL numbers $|u_{\max}|\frac{\Delta t}{\Delta x}$. As the time integration proceeds, a shock wave forms at $x = 0.5$. The shock wave dissipates the solution, decreases the maximum wavespeed $|u|_{\max}$, and increases the size of each time step due to fixed CFL number. As a result, the number of time steps needed to proceed to $t = 1$ is not known *a priori*. To vary the number of time steps, we chose five different CFL numbers ranging from 1.0 to 0.002. The discrete adjoint equation of the original system is solved with initial and boundary conditions,

$$\phi|_{t=1} = \sin 2\pi x, \quad \phi|_{x=0,1} = 0.$$

Four different numbers of checkpoints, $\delta = 10, 25, 50$ and 100, were specified. For each δ, we ran five calculations with different CFL numbers. We recorded the ratio of computation time between the backward sweep of solving adjoint system and the forward sweep of solving the

original system. Because the computation time of the forward sweep is the cost of solving the original system alone, this ratio reflects the additional cost of solving the adjoint equation. We compare the ratio with its theoretical bound defined by Equations (3.3) and (3.2) and by assuming that solving an adjoint step costs the same amount of computation time as a forward step.

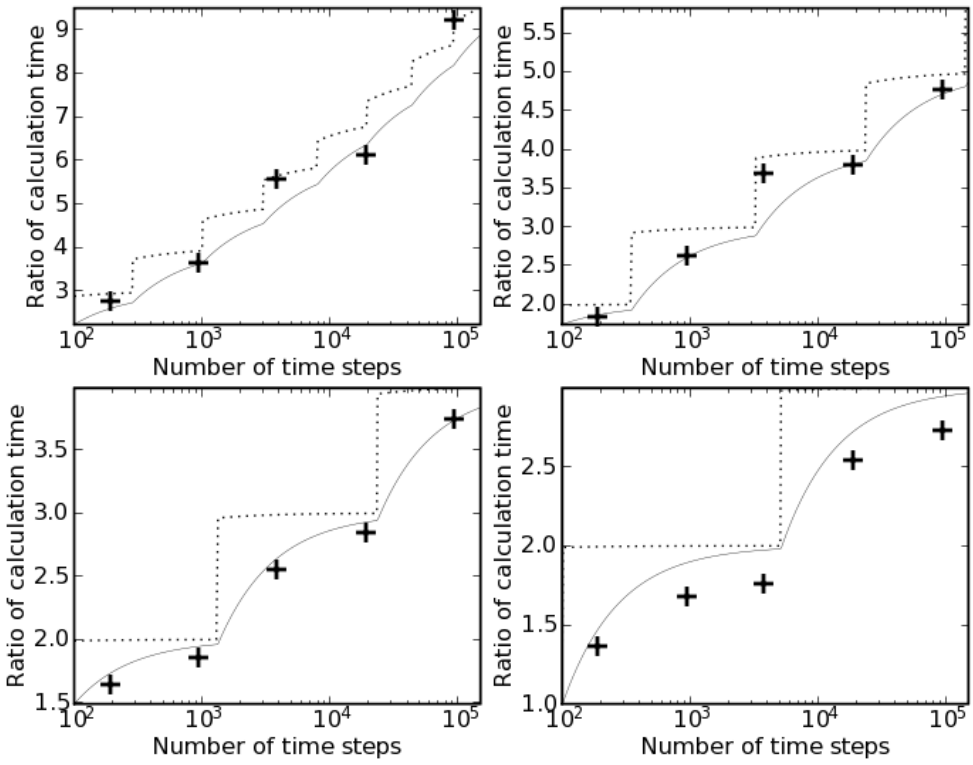

Figure 3.5: Comparison of theoretical bounds (dot and solid lines) and experimental performance (plus markers) of our dynamic checkpointing adjoint solver. The top-left, top-right, bottom-left and bottom-right plots correspond to $\delta = 10, 25, 50$ and 100, respectively.

Figure 3.5 plots this experimental time ratio with the theoretical bounds. The four sub-plots correspond to different number of checkpoints used. Each sub-plot shows the resulting number of time steps and ratio of computation time for 5 different CFL numbers. As can be seen, most of the experimental time ratios are within or close to the theoretical bounds, indicating that our algorithm works in practice as efficiently as theoretically proven. In

this experiment, the computational cost of calculating a linear adjoint step may be smaller than solving a nonlinear Burgers' step, which can explain why some points lies below the theoretical lower bound.

3.5 Conclusion and discussion

We have proposed a checkpointing adjoint solver, including an algorithm for dynamically allocating checkpoints during the initial calculation of the original system and subsequent recalculations. Its three main advantages over previous algorithms are: the number of time steps does not need to be known beforehand; the number of recalculations is minimized; an arbitrary number of time steps can be integrated. For an original system with no more than $\binom{\delta+\tau}{\tau}$ time steps, each time step is calculated at most τ times, as has been proven optimal in the previous literature on checkpointing schemes (Griewank, 1992). Despite the lengthy proof of this optimality, the algorithm itself is conceptually simple to implement and has widespread applications in scientific and engineering simulations of complex systems, where adaptive time-stepping is often desirable, if not necessary.

Although this chapter is biased towards solving adjoint equations of time dependent differential equations, a more compelling application of our algorithm is in reverse mode automatic differentiation. Most scientific computation code contains "if" and "while" statements, making their length of execution uncertain *a priori*. Therefore, our dynamic checkpointing algorithm can be more suitable than static optimal checkpointing algorithms in these cases.

Although we proved that our dynamic checkpointing algorithm has the optimal repetition number, it is not always optimal in terms of the total number of time step recalculations. When the number of time step is between $\binom{\delta+2}{2}$ and $\binom{\delta+3}{3}$, the improved online checkpointing scheme (Stumm & Walther, 2008) may outperform our algorithm. Therefore, our dynamic checkpointing algorithm can still be improved in terms of the total number of recalculations by placing and replacing low level checkpoints in a more planned manner. One may also combine our algorithm with the improved online checkpointing scheme by using online checkpointing until the number of time steps reaches $\binom{\delta+3}{3}$, and switch to our algorithm to proceed further.

Our dynamic checkpointing algorithm aims only to optimize the repetition number and reduce the number of time step calculations, ignoring the computational cost of writing

and reading checkpoints. In many applications, such as solving the incompressible Navier-Stokes equation where each time step involves solving a full Poisson equation, the cost of reading and writing checkpoints is negligible because calculating each time step takes much more computation time. In other cases, especially when the checkpoints are written in and read from hard disk instead of being kept in RAM, the I/O time associated with checkpoint writing and reading cannot be ignored. In such cases, our dynamic checkpointing algorithm may not be the best choice, since it requires a lot more checkpoint writing than **revolve** does.

An implicit assumption of our algorithm is the uniform cost of calculating each time step of the original system. Although this is true for the majority of simple partial differential equations, it is not true for some multi-physics simulations. Moreover, this assumption is false in some automatic differentiation applications. While extension of this algorithm to account for the non-uniform cost of each time step should not be very difficult, maintaining provable performance bound in the extension is subject to further investigation.

Another assumption on which we base our algorithm is that calculating the adjoint solution at time step i from time step $i + 1$ only requires the solution to the original system at time step i. In practice, especially if advanced time integration methods are used in solving the original equation, solving the discrete adjoint equation at time step i may require more than one time step of the original system. Future research plans include investigating ways to adjust our algorithm for this case.

Chapter 4

An Unsteady Adjoint Solver for the Navier-Stokes Equations

4.1 Mathematical formulation of the adjoint equations

This section derives the adjoint equation and the adjoint sensitivity gradient by linearizing the objective functions and the constraints. Section 4.1.1 treats the most general scenario, section 4.1.2 discusses the case when the constraint is a time-dependent equation, and section 4.1.3 derives the adjoint equation and sensitivity gradient when the constraint is the unsteady Navier-Stokes equations, and the objective function is a time-averaged aerodynamic quantity.

4.1.1 General adjoint sensitivity gradient

Let ξ in a Hilbert space Φ be the random variables that describes the sources of uncertainty. Define the objective function as

$$\mathbf{J}(v, \xi) \in R.$$

The state v is in a Banach space $\mathbf{V}$ and satisfies the constraint

$$\mathbf{G}(v, \xi) = 0,$$

where $\mathbf{G} : \mathbf{V} \times \mathbf{\Phi} \rightarrow \mathbf{V}$ is a general nonlinear operator. The constraint uniquely determines a state v for any given control variable ξ.

Let $\xi_0 \in \Phi$ and $v_0 \in \mathbf{V}$ satisfy the constraint

$$\mathbf{G}(v_0, \xi_0) = 0.$$

Let $\xi \in \Phi$ and $v \in \mathbf{V}$ also satisfy the constraint, and both $\delta\xi = \xi - \xi_0$ and $\delta v = v - v_0$ be infinitesimal. Then $\delta\xi$ and δv satisfy the linearized constraint

$$\mathbf{L}\,\delta v + \mathbf{M}\delta\xi = \mathbf{G}(v_0 + \delta v, \xi_0 + \delta\xi) - \mathbf{G}(v_0, \xi_0) = 0$$

where linear operators $\mathbf{L} : \mathbf{V} \to \mathbf{V}$ and $\mathbf{M} : \Phi \to \mathbf{V}$ are defined as

$$\mathbf{L} = \frac{\partial \mathbf{G}}{\partial v}, \quad \mathbf{M} = \frac{\partial \mathbf{G}}{\partial \xi} \quad \text{at } (v_0, \xi_0).$$

On the other hand, the objective function can be linearized as

$$\mathbf{J}(v, \xi) = \mathbf{J}(v_0, \xi_0) + \langle \nabla_\xi \mathbf{J},\ \delta\xi \rangle + \langle \nabla_v \mathbf{J},\ \delta v \rangle$$

$\nabla_\xi \mathbf{J} \in \mathbf{V}^*$ and $\nabla_v \mathbf{J} \in \Phi$ are Frechet derivatives of $\mathbf{J}$ at the linearizing point (ξ_0, v_0).

Now let $\hat{v} \in \mathbf{V}^*$ satisfy the adjoint equation

$$\mathbf{L}^* \hat{v} = \nabla_v \mathbf{J},$$

where $\mathbf{L}^*$ is the adjoint operator of the linear operator $\mathbf{L}$, then

$$\langle \nabla_v \mathbf{J},\ \delta v \rangle = \langle \mathbf{L}^* \hat{v},\ \delta v \rangle = \langle \hat{v},\ \mathbf{L}\,\delta v \rangle = -\langle \hat{v},\ \mathbf{M}\delta\xi \rangle = -\langle \mathbf{M}^* \hat{v},\ \delta\xi \rangle$$

and the objective function can be written as

$$\mathbf{J}(v, \xi) = \mathbf{J}(v_0, \xi_0) + \langle \nabla_\xi \mathbf{J} - \mathbf{M}^* \hat{v},\ \delta\xi \rangle\ .$$

Using this adjoint method, the sensitivity gradient of the objective function becomes

$$\left.\frac{D\mathbf{J}}{D\xi}\right|_{\xi_0} = \nabla_\xi \mathbf{J} - \mathbf{M}^* \hat{v}\ , \tag{4.1}$$

which can be calculated by one adjoint solution.

4.1.2 Adjoint sensitivity gradient for unsteady problems

In unsteady problems, the state variable v has an additional time dimension. i.e., the space of all possible v's can be written as

$$\mathbf{V} = \mathcal{V}^{[0,T]} ,$$

where $\mathcal{V}$ is a Banach space. We call it the "spatial" dimension, as opposed to the temporal dimension. We denote $v(t) \in \mathcal{V}$ as the state at time t. The norm of v in $\mathbf{V}$ should be defined as the supremum of the norm of $v(t)$ in $\mathcal{V}$ over $0 \le t \le T$. The constraint in an unsteady problem is a set of time-dependent ordinary or partial differential equations

$$\mathbf{G}(v, \xi) = \frac{\partial v(t)}{\partial t} + G(v(t), \xi) = 0, \quad 0 \le t \le T .$$

In case of a partial differential equation, G is a spatial operator on $\mathcal{V}$, while ξ is a parameter of the operator. On the other hand, the objective function in unsteady problems can usually be written as

$$\mathbf{J}(v, \xi) = \int_0^T J(v(t), \xi; t)\, dt.$$

In the rest of this chapter, $v(t)$ is simply denoted by the abbreviation v for short where no confusion could result. Similarly, $v_{0\,t}$ and $\delta v(t)$ are denoted as v_0 and δv.

Let ξ_0 and v_0 satisfy the constraint

$$\mathbf{G}(v_0, \xi_0) = 0,$$

and both $\delta v = v - v_0$ and $\delta\xi = \xi - \xi_0$ are infinitesimal. Linearization of the constraint $\mathbf{G}$ around ξ_0, v_0 gives

$$\mathbf{G}(v, \xi) - \mathbf{G}(v_0, \xi_0) = \frac{\partial\, \delta v}{\partial t} + L(t)\delta v + M(t)\delta\xi = 0 , \tag{4.2}$$

where spatial linear operators $L(t) : \mathcal{V} \to \mathcal{V}$ and $M(t) : \Phi \to \mathcal{V}$ are Frechet derivatives defined as

$$L(t) = \frac{\partial G(v, \xi; t)}{\partial v}, \quad M(t) = \frac{\partial G(v, \xi; t)}{\partial \xi} \quad \text{at } (v_0, \xi_0) .$$

These operators may depend on time. On the other hand, linearization of the objective

function $\mathbf{J}$ gives

$$\mathbf{J}(v,\xi) = \mathbf{J}(v_0,\xi_0) + \int_0^T \left(\langle \nabla_\xi J, \delta\xi\rangle + \langle \nabla_v J, \delta v\rangle\right) dt \ , \tag{4.3}$$

where $\nabla_\xi J \in \Phi$ and $\nabla_v J \in \mathcal{V}^*$ are Frechet derivatives of J at (ξ_0, v_0), and the brackets here are spatial inner products, i.e., inner products between spaces $\mathcal{V}^*$ and $\mathcal{V}$.

Now let $\hat{v}$ satisfy the adjoint equation

$$-\frac{\partial \hat{v}}{\partial t} + L^*\hat{v} = \nabla_v J \ .$$

Then

$$\int_0^T \langle \nabla_v J, \delta v\rangle \ dt = -\int_0^T \langle \frac{\partial \hat{v}}{\partial t}, \delta v\rangle \ dt + \int_0^T \langle L^*\hat{v}, \delta v\rangle \ dt \ ,$$

where

$$-\int_0^T \langle \frac{\partial \hat{v}}{\partial t}, \delta v\rangle \ dt = \int_0^T \langle \hat{v}, \frac{\partial \, \delta v}{\partial t}\rangle \ dt$$

through integration by parts, and

$$\int_0^T \langle L^*\hat{v}, \delta v\rangle \ dt = \int_0^T \langle \hat{v}, L\,\delta v\rangle \ dt$$

through the definition of adjoint operator. By combining the above three equations and incorporating Equation (4.2), we obtain

$$\int_0^T \langle \nabla_v J, \delta v\rangle \ dt = \int_0^T \langle \hat{v}, \frac{\partial \, \delta v}{\partial t} + L\,\delta v\rangle \ dt = -\int_0^T \langle \hat{v}, M\,\delta\xi\rangle \ dt = -\int_0^T \langle M^*\hat{v}, \delta\xi\rangle \ dt \ .$$

Incorporating this result into Equation (4.3), the objective function can be obtained as

$$\mathbf{J}(v,\xi) = \mathbf{J}(v_0,\xi_0) + \int_0^T \langle \nabla_\xi J - M^*\hat{v}, \delta\xi\rangle \ dt \ .$$

Therefore, the sensitivity gradient at ξ_0 can be calculated by

$$\left.\frac{D\mathbf{J}}{D\xi}\right|_{\xi_0} = \int_0^T \left(\nabla_\xi J - M_t^*\hat{v}\right) dt \ . \tag{4.4}$$

Calculating the sensitivity gradient using this equation can be done by adding the contribution from each time step of the adjoint solution.

4.1.3 Adjoint analysis for incompressible Navier-Stokes flow

In this subsection, we look at deriving the adjoint sensitivity gradient for uncertainty quantification problems related to incompressible Navier-Stokes flow. The uncertainties affect both the boundaries and the flow in the interior. The former describe The former describes uncertain wall roughness, unmodeled object movement / oscillations etc. The latter describe numerical uncertainties and subgrid model uncertainties. The objective function is assumed to be a measurable aerodynamic quantity, such as lift, drag or moment of a solid object in the flowfield, which depends on the pressure and viscous stress on the boundary of the solid object. The adjoint equations for Navier-Stokes flows have been derived and solved by Jameson (1988), Bewley *et al.* (2001) and Abergel & Temam (1990).

The incompressible unsteady Navier-Stokes and continuity equations with normalized density are

$$\begin{aligned} &\frac{\partial v}{\partial t} + v \cdot \nabla v - \nabla \cdot (\mu \nabla v) + \nabla p = f(\xi) \\ &\nabla \cdot v = 0 \ , \end{aligned}$$

in the spatial domain Ω and time period $[0, T]$. The source term $f(\xi)$ depends on ξ, the random variables describing the sources of uncertainties. The initial condition is

$$v = u_0, \quad t = 0$$

and the boundary condition are dependent on the sources of uncertainties

$$v = v_b(\xi), \quad x \in \partial\Omega.$$

The objective function is a time-integrated aerodynamic quantity, which depends on the pressure and viscous stresses at the boundary:

$$\mathbf{J}(v, \xi) = \int_0^T J(\mathbf{n} \cdot \tau \,|_{\partial\Omega}, \, p \,|_{\partial\Omega}; \, t) \, dt \ ,$$

where the viscous stress tensor is $\tau = \mu \nabla v$, and $\mathbf{n}$ is the unit wall-normal.

Let ξ_0 be the point of linearization, v_0 and p_0 be the solution of the Navier-Stokes equations with boundary condition $v_b(\xi_0)$ and forcing term $f(\xi_0)$. We can linearize both the Navier-Stokes equations and the objective function. By defining $\delta\xi = \xi - \xi_0$, $\delta v = v - v_0$

and $\delta p = p - p_0$, the linearized incompressible LES equation is

$$\begin{aligned} &\frac{\partial\, \delta v}{\partial t} + \mathcal{L}_{v_0}\, \delta v + \nabla \delta p = \frac{\partial f}{\partial \xi}\, \delta \xi \\ &\nabla \cdot \delta v = 0 \end{aligned} \tag{4.5}$$

with boundary condition

$$\delta v = \frac{\partial v_b}{\partial \xi}\, \delta \xi, \quad x \in \partial\Omega,$$

and initial condition

$$\delta v = 0, \quad t = 0.$$

The linearized Navier-Stokes operator $\mathcal{L}_{v_0}$ is defined as

$$\mathcal{L}_{v_0} \delta v = \delta v \cdot \nabla v_0 + v_0 \cdot \nabla \delta v - \nabla \cdot (\mu \nabla \delta v)\ .$$

Note that the operator depends on the point of linearization v_0.

On the other hand, the linearized objective function is

$$\begin{aligned} \delta \mathbf{J}(v, \xi) &= \mathbf{J}(v, \xi) - \mathbf{J}(v_0, \xi_0) \\ &= \int_0^T \iint_{\partial\Omega} (a(x,t) \cdot (\mathbf{n} \cdot \delta\tau) + b(x,t)\ \delta p)\, ds\ dt \\ &+ \int_0^T \iiint_{\Omega} c(x,t) \cdot \delta v\ dx\ dt\ . \end{aligned} \tag{4.6}$$

where

$$\delta\tau = \tau - \tau_0 = \mu_0 \nabla \delta v + \delta\mu \nabla v_0$$

$a(x,t)$, $b(x,t)$ and $c(x,t)$ are the Frechet derivatives

$$a = \frac{\partial J}{\partial\, (\mathbf{n} \cdot \tau)}\ , \qquad b = \frac{\partial J}{\partial p}\ , \qquad c = \frac{\partial J}{\partial v}$$

a is a vector function at the boundaries, b is a scalar function at the boundaries, and c is a vector function in the interior.

Define the adjoint variables $\hat{v}$, $\hat{p}$ and $\hat{\mu}$ such that they satisfy the adjoint equation

$$\begin{aligned} -\frac{\partial \hat{v}}{\partial t} + \mathcal{L}^*_{v_0}\hat{v} + \nabla\hat{p} &= -c \\ \nabla\cdot\hat{v} &= 0 \end{aligned} \tag{4.7}$$

where $\mathcal{L}^*_{v_0}$ is the adjoint operator of $\mathcal{L}_{v_0}$:

$$\mathcal{L}^*_{v_0}\hat{v} = \nabla v_0\cdot\hat{v} - v_0\cdot\nabla\hat{v} - \nabla\cdot(\mu_0\nabla\hat{v})\ .$$

The adjoint variables also must satisfy the terminal condition

$$\hat{v} = 0, \quad t = T$$

and the adjoint boundary condition

$$\hat{v} = a - \mathbf{n}\,(b + a\cdot\mathbf{n})\,, \quad x\in\partial\Omega\ . \tag{4.8}$$

We now show that the sensitivity derivatives of the objective function $\mathbf{J}$ with respect to ξ can be calculated using ξ and adjoint variables.

First, the adjoint boundary condition (4.8) implies that

$$\delta p\ \mathbf{n}\cdot\hat{v} = b\,\delta p \tag{4.9}$$

and because $\mathbf{n}\cdot\tau\cdot\mathbf{n}\equiv 0$ on the boundaries, (4.8) further implies

$$\mathbf{n}\cdot(\mu\nabla\delta v)\cdot\hat{v} = \mathbf{n}\cdot\delta\tau\cdot\hat{v} = -\mathbf{n}\cdot\delta\tau\cdot a\ . \tag{4.10}$$

In addition, from the adjoint equation,

$$c\cdot\delta v = -\delta v\cdot\left(-\frac{\partial\hat{v}}{\partial t} + \mathcal{L}^*_{v_0}\hat{v} + \nabla\hat{p}\right) \tag{4.11}$$

and

$$0 = \delta p\ \nabla\cdot\hat{v} \tag{4.12}$$

Incorporating (4.9), (4.10), (4.11) and (4.12) into the linearized objective function (4.6), we

get

$$\begin{aligned}\delta\mathbf{J}(v,\xi) = &\int_0^T \iint_{\partial\Omega} (\mathbf{n}\cdot(\mu\nabla\delta v)\cdot\hat{v} - \delta p\,\mathbf{n}\cdot\hat{v})\,ds\,dt \\ &- \int_0^T \iiint_{\Omega} \delta v\cdot\left(-\frac{\partial\hat{v}}{\partial t} + \mathcal{L}^*_{v_0}\hat{v} + \nabla\hat{p}\right)dx\,dt \\ &+ \int_0^T \iiint_{\Omega} \delta p\,\nabla\cdot\hat{v}\,dx\,dt\end{aligned} \tag{4.13}$$

Furthermore, using integration by parts, and applying the initial and boundary conditions for both the linearized Navier-Stokes equations and the adjoint equation, we get the following four equalities:

$$\int_0^T \hat{v}\cdot\frac{\partial\,\delta v}{\partial t}dt = -\int_0^T \delta v\cdot\frac{\partial\hat{v}}{\partial t}dt \tag{4.14}$$

$$\begin{aligned}\iiint_{\Omega} \hat{v}\cdot\mathcal{L}_{v_0}\delta v\,dx = &\iiint_{\Omega} \delta v\cdot\mathcal{L}^*_{v_0}\hat{v}\,dx \\ &- \iint_{\partial\Omega} \mathbf{n}\cdot(\mu\nabla\delta v)\cdot\hat{v}\,ds \\ &+ \iint_{\partial\Omega} \mathbf{n}\cdot(\mu\nabla\hat{v} - v_w\hat{v})\cdot\delta v\,ds\end{aligned} \tag{4.15}$$

$$\iiint_{\Omega} \hat{v}\cdot\nabla\delta p\,dx = -\iiint_{\Omega} \delta p\,\nabla\cdot\hat{v}\,dx + \iint_{\partial\Omega} \delta p\,\mathbf{n}\cdot\hat{v}\,ds \tag{4.16}$$

$$\iiint_{\Omega} \hat{p}\,\nabla\cdot\delta v\,dx = -\iiint_{\Omega} \delta v\cdot\nabla\hat{p}\,dx + \iint_{\partial\Omega} \hat{p}\,\mathbf{n}\cdot\delta v\,ds\,. \tag{4.17}$$

Incorporating these four equations (4.9), (4.10), (4.11) and (4.12) into (4.13), we get

$$\begin{aligned}\delta\mathbf{J}(v,\xi) = &\int_0^T \iint_{\partial\Omega} (\mathbf{n}\cdot(\mu\nabla\hat{v}) - \mathbf{n}\cdot v_w\,\hat{v} - \hat{p}\,\mathbf{n})\cdot\delta v\,ds \\ &- \int_0^T \iiint_{\Omega} \hat{v}\cdot\left(\frac{\partial\,\delta v}{\partial t} + \mathcal{L}_{v_0}\delta v + \nabla\delta p\right)dx\,dt \\ &+ \int_0^T \iiint_{\Omega} \hat{p}\,\nabla\cdot\delta v\,dx\,dt \\ = &\int_0^T \iint_{\partial\Omega} (\mathbf{n}\cdot(\mu\nabla\hat{v}) - \mathbf{n}\cdot v_w\,\hat{v} - \hat{p}\,\mathbf{n})\cdot\frac{\partial v_b}{\partial\xi}\,\delta\xi\,ds \\ &- \int_0^T \iiint_{\Omega} \hat{v}\cdot\frac{\partial f}{\partial\xi}\,\delta\xi\,dx\,dt\,.\end{aligned}$$

Therefore, the sensitivity gradient is

$$\begin{aligned}\frac{D\mathbf{J}}{D\xi} = &\int_0^T \iint_{\partial\Omega} (\mathbf{n}\cdot(\mu\nabla\hat{v}) - \mathbf{n}\cdot v_w\,\hat{v} - \hat{p}\,\mathbf{n})\cdot\frac{\partial v_b}{\partial\xi}\,ds \\ &- \int_0^T \iiint_{\Omega} \hat{v}\cdot\frac{\partial f}{\partial\xi}\,dx\,dt\,.\end{aligned} \tag{4.18}$$

It can be calculated by using the adjoint solution at each time step. This is a key motivation for obtaining the adjoint solution in the first place.

4.2 The adjoint Navier-Stokes solver

This section describes the numerical scheme used to solve the adjoint Navier-Stokes equations. In designing the numerical scheme, we follow the "discrete adjoint" approach. The numerical scheme we obtain by this approach is not only a consistent discretization of the continuous adjoint Navier-Stokes equations, it is also the direct discrete adjoint of the numerical scheme for solving the unsteady Navier-Stokes equations. This approach has an important advantage[1]: the sensitivity gradient we obtained is not affected by discretization error of either the Navier-Stokes equation or the adjoint Navier-Stokes equations. Since the discrete adjoint scheme is dependent on the scheme for solving the Navier-Stokes equations, we start by describing CDP, our unsteady Navier-Stokes solver on an unstructured hybrid mesh.

4.2.1 The scheme for solving the unsteady Navier-Stokes equations

The 3-D unsteady Navier-Stokes solver is called CDP, and is developed and maintained by Dr. Frank Ham of the Center for Turbulence Research (Ham *et al.*, 2007). The solver uses a fractional-step method to enforce the divergence-free condition. In the most recent version of the solver, an implicit BDF-2 time integration scheme (Hosea & Shampine, 1996) is used for time discretization. The spatial discretization is a node based finite volume method. Velocity and pressure are stored in the dual-volume corresponding to each node. They are denoted as u without subscript, and p, respectively. A separate velocity field is stored on the surfaces of the dual-volumes, precisely corresponding to the face edges, to enforce the divergence-free condition. This auxiliary velocity field is denoted as u_m. The

[1]The discrete adjoint also have disadvantages compared to the continuous adjoint, such as its dependency on the primal scheme, and generally greater complexity.

Simplex Superposition scheme (Ham, 2008) recently developed by Dr. Ham is used for spatial discretization of the gradient, divergence, Laplacian and convection operators. The scheme is second-order accurate both in space and time for arbitrarily shaped elements.

A mathematical description of the discretization scheme is

$$\begin{aligned}
u_m^{k*} &= 2u_m^{k-1} - u_m^{k-2} \\
u^{k-\frac{2}{3}} &= \frac{4}{3}u^{k-1} - \frac{1}{3}u^{k-2} \\
\frac{3}{2}\frac{u^{k*} - u^{k-\frac{2}{3}}}{\Delta t} &= -u_m^{k*} \cdot \nabla u^{k*} + \nabla \cdot \mu \nabla u^{k*} - \nabla p^{k-1} \\
u^{k-} &= u^{k*} + \frac{2\Delta t}{3}\nabla p^{k-1} \\
\Delta p^k &= \frac{3}{2\Delta t}\nabla \cdot u^{k-} \\
u^k &= u^{k-} - \frac{2\Delta t}{3}\nabla p^k \\
u_m^k &= P_F\, u^{k-} - \frac{2\Delta t}{3}\nabla_F\, p^k \ .
\end{aligned} \tag{4.19}$$

In this formula, the first three lines are the prediction step, including the BDF-2 scheme (the second and the third lines); the last four lines are the pressure correction step. u^{k-1} is the nodal velocity at time step $k-1$. u_m^{k-1} is the auxiliary divergence-free velocity stored on the interface of the dual-volumes at time step $k-1$. u_m^{k*} is the extrapolated auxiliary velocity field based on u_m at time steps $k-1$ and $k-2$, and is used as the convecting velocity in the next prediction step. $u^{k-\frac{2}{3}}$ is an extrapolated nodal velocity field from time step $k-1$ and $k-2$, and is used by the BDF-2 time integration scheme to achieve its formal second-order time accuracy. u^{k*} is the nodal velocity at time step k predicted by the prediction step. It is obtained by time-integrating the full Navier-Stokes equations using the BDF-2 scheme, using the pressure field p^{k-1} at time step $k-1$ instead of the pressure field at time step k. u^k, the final nodal velocity field, is obtained after the correction step. After u^{k*} is calculated, we calculate u^{k-} by subtracting the effect of the pressure field at time step $k-1$ from u^{k*}. The pressure p^k at time step k is then calculated from the divergence of u^{k-}. The nodal gradient of the new pressure field is then added to u^{k-} to obtain u^k, the nodal velocity at time step k. The new auxiliary velocity field, u_m^k, is obtained by transferring u^{k-} to the dual-cell interfaces, then add the interfacial gradient of the new pressure field p^k. The transfer operator P_F and interfacial gradient operator ∇_F are designed so that the divergence of u_m^k is exactly zero. The kth time step is completed when p^k, u^k and u_m^k are

obtained.

Six discrete spatial operators are used in the scheme: the convection operator $u_m \cdot \nabla$, the nodal divergence operator $\nabla \cdot$, the nodal gradient operator ∇, the nodal Laplacian operator Δ, the nodal to facial transfer operator P_F, and the nodal to facial gradient operator ∇_F. In the most recent version of CDP, these discrete operators are constructed with the Simplex Superposition principle (Ham, 2008). These operators satisfy $\nabla_F \cdot \nabla_F = \Delta$ and $\nabla_F \cdot P_F = \nabla \cdot$, ensuring the discrete divergence-free condition of the facial velocity u_m, i.e., $\nabla_F \cdot u_m = 0$. The Simplex-Superposition based schemes have excellent memetic properties (Ham, 2008), in the sense that the discrete operators retain many benign properties of their continuous counterparts, such as discrete conservation of kinetic energy (Morinishi *et al.*, 1998). In addition, due to their memetic properties, the Laplacian operator is discretely self-adjoint, while the nodal divergence operator and the nodal gradient operator are discretely adjoint to each other. These properties makes construction of the discrete adjoint scheme easier.

4.2.2 The homogeneous adjoint scheme

A homogeneous adjoint equation is an adjoint equation with zero source term and trivial boundary conditions. It corresponds to the continuous adjoint equation (4.7) and boundary condition (4.8) with $a = b = c = 0$. The following equations describe the discrete adjoint version of (4.19) that solves the homogeneous adjoint equation.

$$
\begin{aligned}
\Delta q^k &= \frac{\hat{p}^k + \nabla \cdot \hat{u}^k}{\Delta t} + \nabla_F \cdot \hat{u}_m^{k*} \\
\hat{u}^{k-} &= \hat{u}^k + \Delta t \left(P_N \, \hat{u}_m^{k*} - \nabla q^k \right) \\
\frac{3}{2} \frac{\hat{u}^{k-\frac{2}{3}} - \hat{u}^{k-}}{\Delta t} &= u_m^{k*} \cdot \nabla \hat{u}^{k-\frac{2}{3}} + \nabla \cdot \mu \nabla \hat{u}^{k-\frac{2}{3}} \\
\hat{u}^{k-1} &= \frac{4}{3} \hat{u}^{k-\frac{2}{3}} - \frac{1}{3} \hat{u}^{k+\frac{1}{3}} \\
\hat{u}_m^k &= -\frac{2}{3} \nabla u^{k*} \cdot \hat{u}^{k-\frac{2}{3}} \\
\hat{p}^{k-1} &= \nabla \cdot \left(\hat{u}^{k-\frac{2}{3}} - \hat{u}^{k-} \right) \\
\hat{u}_m^{k-1*} &= 2\hat{u}_m^k - \hat{u}_m^{k+1}
\end{aligned}
\tag{4.20}
$$

A brief derivation of this scheme is presented in Appendix A. One can show that this scheme is a consistent (though unusual) discretization of the homogeneous continuous adjoint equation, i.e., equation (4.7), with zero source term c. Therefore, as the mesh is refined, the solution using this scheme converges to the continuous adjoint solution. In addition, this scheme is the discrete adjoint of the scheme (4.19). If we write the discrete tangent linear Navier-Stokes equations as a single matrix equation whose solution consists of its solution at all grid points and all time steps, our discrete adjoint scheme (4.20) would be equivalent to solving the transpose of the huge matrix equation. For this reason, the adjoint solution of this scheme and the Navier-Stokes solution using scheme (4.19) satisfy the discrete adjoint condition[2]:

$$\begin{aligned}
&\sum \left(\delta u^k \cdot \hat{u}^k - \frac{1}{3}\, \delta u^{k-1} \cdot \hat{u}^{k+\frac{1}{3}} \right) dV + \\
&\sum \Delta t \left(\delta u_m^k \cdot \hat{u}_m^{k*} - \delta u_m^{k-1} \cdot \hat{u}_m^{k+1} + \frac{2}{3}\, \delta p^k \hat{p}^k \right) dV \\
=&\sum \left(\delta u^{k-1} \cdot \hat{u}^{k-1} - \frac{1}{3}\, \delta u^{k-2} \cdot \hat{u}^{k-\frac{2}{3}} \right) dV + \\
&\sum \Delta t \left(\delta u_m^{k-1} \cdot \hat{u}_m^{k-1*} - \delta u_m^{k-2} \cdot \hat{u}_m^{k} + \frac{2}{3}\, \delta p^{k-1} \hat{p}^{k-1} \right) dV \ .
\end{aligned}$$

Or equivalently,

$$\begin{aligned}
&\sum \left(\delta u^k \cdot \hat{u}^k - \frac{1}{3}\, \delta u^{k-1} \cdot \hat{u}^{k+\frac{1}{3}} \right) dV + \\
&\sum \Delta t \left(\delta u_m^k \cdot \hat{u}_m^{k*} - \delta u_m^{k-1} \cdot \hat{u}_m^{k+1} + \frac{2}{3}\, \delta p^k \hat{p}^k \right) dV = \text{Constant for all } k \ .
\end{aligned}$$

In this discrete adjoint condition, $\sum dV$ represents a numerical integration of a scalar field over the entire volume. Specifically,

$$\sum \phi \ dV = \sum \phi_i dV_i, \tag{4.21}$$

where the summation is over all the grid points in the mesh, ϕ_i is the value of the scalar field on the ith mesh point, and dV_i is the volume of the corresponding dual-control volume. This notation is also used to describe the inhomogeneous discrete adjoint condition in the next subsection. Δt is the size of the time step between the $k-1$st and the kth time steps.

[2] A rigorous proof of this discrete adjoint condition is presented in Appendix A.

This discrete adjoint condition is satisfied with any mesh resolution and time step.

The adjoint equation evolves backward in time; its terminal condition is set at the last time step of the Navier-Stokes time integration, and each evaluation of the adjoint scheme (4.20) brings the adjoint solution one time step backward. Note that the primal intermediate facial and nodal velocities u_m^{k*} and u^{k*} appear in (4.20), which is consistent with the fact that the continuous form of the adjoint equation (4.7) contains the primal velocity u. This fact suggests that applying the adjoint scheme at the kth time step requires information from the Navier-Stokes solution at the same time step. For this reason, solving the adjoint equation for each time step involves some preparation before performing the calculation in (4.20):

1. Obtain the Navier-Stokes state variables u^{k-1}, u^{k-2}, u_m^{k-1}, u_m^{k-2} and p^{k-1}.
2. Solve the prediction part (first three lines) of the Navier-Stokes step (4.19) for u^{k*} and $u_,^{k*}$.
3. Retrograde the adjoint solution using the adjoint scheme (4.20).

Since the adjoint equation is calculated retrograde in time, the Navier-Stokes state variables are needed in a reverse-time order. The dynamic checkpointing scheme described in Chapter 3 is applied to fulfill this requirement. Using our dynamic checkpointing scheme, the adjoint equation can be solved at a computational cost of around 4 to 8 times that of the Navier-Stokes equations, with moderate increase in memory usage.

4.2.3 The inhomogeneous adjoint scheme and sensitivity gradient

In the previous section, we discussed the scheme for solving homogeneous adjoint equation, i.e., the adjoint equation with a zero source term and trivial boundary conditions. In practice, most objective functions are time integrals of physical quantities in the flowfield or on the boundary. Consequently, the adjoint equations for these objective functions has either a nonzero source term or nontrivial boundary condition. This section focuses on these inhomogeneous adjoint equations. We discuss how to construct the source term based on the specific form of the objective function, and how to calculate the sensitivity gradients using the solution of the adjoint equation.

In general, the objective function depends on the velocity field and the pressure field.

Denote the objective function as $\mathbf{J}$, and let

$$J^k_{u\,i} = \frac{\partial \mathbf{J}}{\partial u^k_i} \ , \quad J^k_{p\,i} = \frac{\partial \mathbf{J}}{\partial p^k_i}$$

the derivatives of the objective function with respect to the velocity and pressure at every grid point and every time step. The general form of the objective function can then be linearized as

$$\delta \mathbf{J} = \sum_{k=0}^{M_T} \Delta t \sum \left(J^k_u \cdot \delta u^k + J^k_p \, \delta p^k \right) dV,$$

where M_T is the last time step in the problem considered, and the inner summation is as defined in Equation (4.21). Similarly, a general assumption with the uncertainties is that they add a source term to the momentum equation anywhere and anytime. In other words, the uncertainties lead to the following modification of the velocity field

$$u^k = u^k + \Delta t \ f^k(\xi)$$

at the end of every time step of the Navier-Stokes equations. Its linearized form is

$$\delta u^k = \delta u^k + \Delta t \ \frac{\partial f^k}{\partial \xi} \ \delta \xi. \tag{4.22}$$

The initial condition can also be uncertain, and is a function of ξ. The linearized initial condition is

$$\delta u^0 = \frac{\partial u^0}{\partial \xi} \ \delta \xi$$
$$\delta u^0_m = \frac{\partial u^0_m}{\partial \xi} \ \delta \xi \ .$$

The form of the inhomogeneous adjoint equation depends on the linearized objective function, specifically on J_u and J_p. On the other hand, the formula for computing the sensitivity gradient from the adjoint solution depends on how the sources of uncertainties influence the system, specifically on $\dfrac{\partial f^k}{\partial \xi}$.

We solve the inhomogeneous adjoint equation with the same scheme as (4.20), except

at the end of each time step, we let

$$\begin{aligned} \hat{u}^{k-1} &= \hat{u}^{k-1} + \Delta t \; J_u^{k-1} \\ \hat{p}^{k-1} &= \hat{p}^{k-1} + \frac{3}{2} \; J_p^{k-1} \; , \end{aligned} \tag{4.23}$$

where the $\frac{3}{2}$ in the adjoint pressure update is due to the BDF-2 scheme. This addition of a source term in the linearized Navier-Stokes equations as in (4.22), and in the adjoint Navier-Stokes equations as in (4.25), both modifying the discrete adjoint condition from its homogeneous form. The inhomogeneous discrete adjoint condition is

$$\begin{aligned} &\sum \left(\delta u^k \cdot \hat{u}^k - \frac{1}{3} \, \delta u^{k-1} \cdot \hat{u}^{k+\frac{1}{3}} \right) dV + \\ &\sum \Delta t \left(\delta u_m^k \cdot \hat{u}_m^{k*} - \delta u_m^{k-1} \cdot \hat{u}_m^{k+1} + \frac{2}{3} \, \delta p^k \hat{p}^k \right) dV - \Delta t \sum \left(\frac{\partial f^k}{\partial \xi} \cdot \hat{u}^k \; \delta \xi \right) dV \\ = &\sum \left(\delta u^{k-1} \cdot \hat{u}^{k-1} - \frac{1}{3} \, \delta u^{k-2} \cdot \hat{u}^{k-\frac{2}{3}} \right) dV + \\ &\sum \Delta t \left(\delta u_m^{k-1} \cdot \hat{u}_m^{k-1*} - \delta u_m^{k-2} \cdot \hat{u}_m^{k} + \frac{2}{3} \, \delta p^{k-1} \hat{p}^{k-1} \right) dV - \\ &\Delta t \sum \left(\delta u^{k-1} \cdot J_u^{k-1} + \delta p^{k-1} J_p^{k-1} \right) dV \; , \end{aligned}$$

where the summation is as defined in Equation (4.21). Applying this equality for $k = 1, 2, \ldots, M_T$, we get

$$\begin{aligned} &\sum \left(\delta u^{M_T} \cdot \hat{u}^{M_T} - \frac{1}{3} \, \delta u^{M_T-1} \cdot \hat{u}^{M_T+\frac{1}{3}} \right) dV + \\ &\sum \Delta t \left(\delta u_m^{M_T} \cdot \hat{u}_m^{M_T*} - \delta u_m^{M_T-1} \cdot \hat{u}_m^{M_T+1} + \frac{2}{3} \, \delta p^{M_T} \hat{p}^{M_T} \right) dV + \\ &\sum_{k=0}^{M_T-1} \Delta t \sum \left(\delta u^k \cdot J_u^k + \delta p^k J_p^k \right) dV \\ = &\sum \left(\delta u^0 \cdot \hat{u}^0 - \frac{1}{3} \, \delta u^{-1} \cdot \hat{u}^{\frac{1}{3}} \right) dV + \\ &\sum \Delta t \left(\delta u_m^0 \cdot \hat{u}_m^{0*} - \delta u_m^{-1} \cdot \hat{u}_m^{1} + \frac{2}{3} \, \delta p^0 \hat{p}^0 \right) dV + \\ &\sum_{k=1}^{M_T} \Delta t \sum \left(\frac{\partial f^k}{\partial \xi} \cdot \hat{u}^k \; \delta \xi \right) dV \; . \end{aligned} \tag{4.24}$$

When the simulation starts, u^{-1}, u_m^{-1} and p^0 are set to 0. So there are no uncertainties in these quantities. Also, we set the adjoint terminal condition as

$$\begin{aligned}
\hat{u}^{M_T} &= \Delta t \; J_u^{k-1} \\
\hat{p}^{M_T} &= \frac{3}{2} \; J_p^{k-1} \\
\hat{u}^{M_T+\frac{1}{3}} &= 0 \\
\hat{u}_m^{M_T} &= 0 \\
\hat{u}_m^{M_T+1} &= 0 \; .
\end{aligned} \tag{4.25}$$

The inhomogeneous discrete adjoint condition (4.24) becomes

$$\begin{aligned}
&\sum_{k=0}^{M_T} \Delta t \sum \left(\delta u^k \cdot J_u^k + \delta p^k J_p^k \right) dV \\
=&\sum \left(\delta u^0 \cdot \hat{u}^0 + \Delta t \; \delta u_m^0 \cdot \hat{u}_m^{0*} \right) dV + \sum_{k=1}^{M_T} \Delta t \sum \left(\frac{\partial f^k}{\partial \xi} \cdot \hat{u}^k \; \delta \xi \right) dV \; .
\end{aligned}$$

In other words,

$$\delta \mathbf{J} = \sum \left(\frac{\partial u^0}{\partial \xi} \cdot \hat{u}^0 + \Delta t \; \frac{\partial u_m^0}{\partial \xi} \cdot \hat{u}_m^{0*} \right) \delta \xi \; dV + \sum_{k=1}^{M_T} \Delta t \sum \left(\frac{\partial f^k}{\partial \xi} \cdot \hat{u}^k \; \delta \xi \right) dV \; .$$

Therefore, the sensitivity gradient can be calculated by

$$\frac{D\mathbf{J}}{D\xi} = \sum \left(\frac{\partial u^0}{\partial \xi} \cdot \hat{u}^0 + \Delta t \; \frac{\partial u_m^0}{\partial \xi} \cdot \hat{u}_m^{0*} \right) \; dV + \sum_{k=1}^{M_T} \Delta t \sum \left(\frac{\partial f^k}{\partial \xi} \cdot \hat{u}^k \right) dV \; . \tag{4.26}$$

We note that the adjoint solution at all time steps is used in calculating the sensitivity gradient. After the adjoint solution is computed at each time step k, $\sum \left(\frac{\partial f^k}{\partial \xi} \cdot \hat{u}^k \right) dV$ is calculated and added to the sensitivity gradient. The full sensitivity gradient can be computed only after the adjoint solution reaches time step 0.

4.3 Numerical example of an adjoint solution

The last section discussed how to solve the adjoint Navier-Stokes equations for general objective functions. This section gives an example of the process. We consider a Newtonian

incompressible fluid flowing past an infinitely long circular cylinder. The Reynolds number with respect to the far-field flow velocity and the cylinder diameter is 100. At this Reynolds number, the flowfield is 2-D, unsteady and periodic in time (Fey *et al.*, 1998).

The incompressible Navier-Stokes equations were solved in a 2-D domain of size 60 cylinder diameters (flow direction) by 80 cylinder diameters (crossflow direction). The domain was discretized with approximately 10,000 unstructured quadrangle mesh elements. By taking advantage of the unstructured grid, most of the mesh points were concentrated around the cylinder and in a band of length 10 cylinder diameters downstream of the cylinder. Figure 4.1 shows the unstructured mesh used. The flow velocity on the left

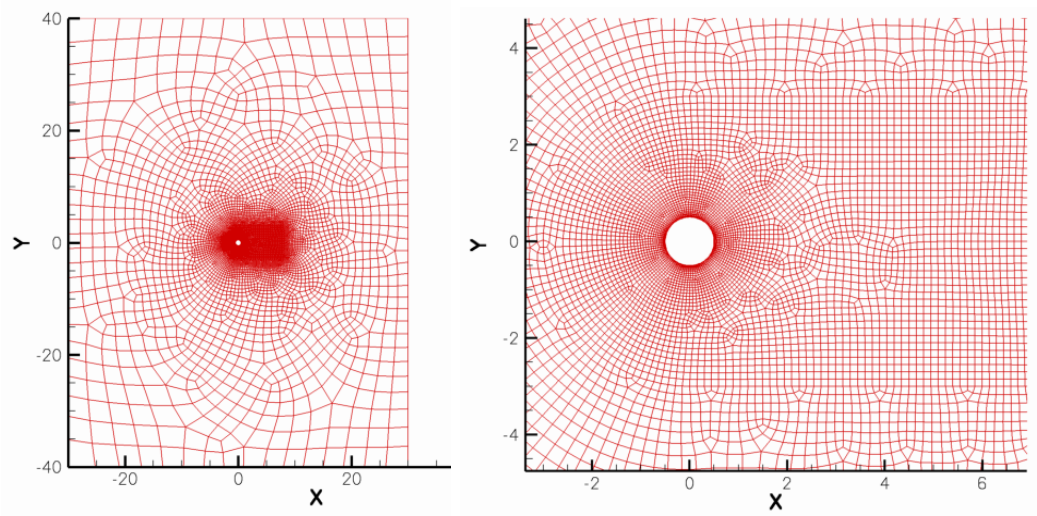

Figure 4.1: The mesh used for calculating the flowfield at Reynolds number 100. The left picture shows the entire computational domain of 60×80; the right picture zooms in to a small region around the cylinder.

boundary of the domain was set to the freestream velocity, the upper and lower boundaries of the domain were set to periodic, and an convective outlet boundary condition was used at the downstream boundary. An algebraic multigrid solver was used to solve the Poisson equation in the corrector steps. We used a fixed time step size, $\Delta t = 0.1$, in this calculation. The corresponding CFL number was approximately 3 in the region near the cylinder. All physical quantities were normalized with respect to the fluid density, the freestream velocity and the cylinder diameter.

The uncertainty in the problem we consider comes from small rotational oscillations of the cylinder in the flow. The specific form of the oscillation is unknown, and the rate of

rotation ω is modeled as a random time process. A no-slip boundary condition is used on the wall, which is

$$v_x = \omega y, \quad v_y = -\omega x.$$

Oscillatory rotation of the cylinder has been shown to have significant effect on the drag coefficient (Tokumaru & Dimotakis, 2006). The objective of this example is to analyze the effect of the small random rotational oscillations on the drag coefficient. Specifically, we want to obtain the probability distribution of the time-averaged drag coefficient of the cylinder from the random rotation. We describe this process as ''propagating" the uncertainties from the sources, the random rotation in this example, to the objective quantity, the time-averaged drag coefficient $\bar{c_d}$.

The objective function is

$$\mathbf{J} = \overline{c_d} = \frac{1}{T}\int_0^T c_d \, dt \ ,$$

where c_d is the instantaneous drag coefficient,

$$c_d = \frac{D(t)}{\frac{1}{2}\rho v_\infty^2 d} \ .$$

$D(t)$ is the instantaneous drag on the cylinder per spanwise length, ρ is fluid density, v_∞ is freestream velocity and d is the cylinder diameter. Since all quantities are normalized with ρ, v_∞ and d, $\rho v_\infty^2 d = 1$, and

$$c_d = 2D(t) \ .$$

The instantaneous unit-span drag $D(t)$ consists of pressure drag D_p and viscous drag D_v, which are

$$D_p = -\oint p\mathbf{n} \cdot \mathbf{e_x} ds$$

$$D_v = \oint \mathbf{n} \cdot \tau \cdot \mathbf{e_x} ds$$

and

$$c_d = 2\oint (\mathbf{n} \cdot \tau - p\mathbf{n}) \cdot \mathbf{e_x} \, ds$$

where $\mathbf{n}$ is wall-normal unit vector, μ is viscosity, $\mathbf{e_x} = (1, 0, 0)$ is the unit vector in the x-direction, and the integration is on the circumference of the cylinder cross-section.

A numerical approximation of the time-averaged drag coefficient is

$$\mathbf{J} = \frac{2}{T}\sum_{k=1}^{T}\Delta t \sum_{i\in sfc}(\mathbf{n}_i \cdot \tau_i - p_i\mathbf{n}_i)\cdot \mathbf{e}_\mathbf{x} ds_i \ . \tag{4.27}$$

In the formula, sfc represents mesh nodes on the wall boundary of the cylinder, p_i is the pressure, and τ_i is the viscous stress at node i, which can be directly calculated from the velocity of its neighboring nodes:

$$\tau_i = \frac{1}{Re}\nabla v|_i = \frac{1}{Re}\sum_{j\in nbr(i)} \kappa_{i\,j} v_j \ ,$$

where $\kappa_{i\,j}$ is the Simplex-Superposition discretization of the gradient operator, its specific value depends on the cell types and geometry. We note that this objective function is linear with respect to the velocity and pressure. The source term of the adjoint equation corresponding to this objective function is

$$\begin{aligned} J_{v\,i} &= \sum_{j\in sfc\cup nbr(i)} \mathbf{n}_j \cdot \kappa_{j\,i}\mathbf{e}_\mathbf{x} \\ J_{p\,i} &= \begin{cases} \mathbf{n}_i \cdot \mathbf{e}_\mathbf{x} & i \in sfc \\ 0 & i \notin sfc \end{cases} . \end{aligned} \tag{4.28}$$

We use the numerical schemes (4.20) and (4.25) with the source term (4.28) to solve the inhomogeneous adjoint equation, so that the corresponding objective function is the discretized time-averaged drag coefficient (4.27). Several snapshots of the adjoint solution are depicted in Figure 4.2. The left side of the figure shows the streamwise velocity u_x at four time instances; the right side shows the corresponding adjoint streamwise velocity $\hat{u}_x$. These adjoint fields reveals the sensitivity derivative of the objective function with respect to changes in the flowfield. In these plots, red is positive values of $\hat{u}_x$, indicating that a positive change in the streamwise velocity would increase the drag of the cylinder. Such regions include the upstream of the cylinder, where an increase in u_x would generate more skin friction drag, and unsteady bands in the shear layers on the sides of the cylinder, where an increase in u_x would temporarily widen the wake, creating more pressure drag on the cylinder. In contrast, blue is negative values of $\hat{u}_x$, indicating that a positive change in the streamwise velocity would reduce the drag of the cylinder. These regions are unsteady bands

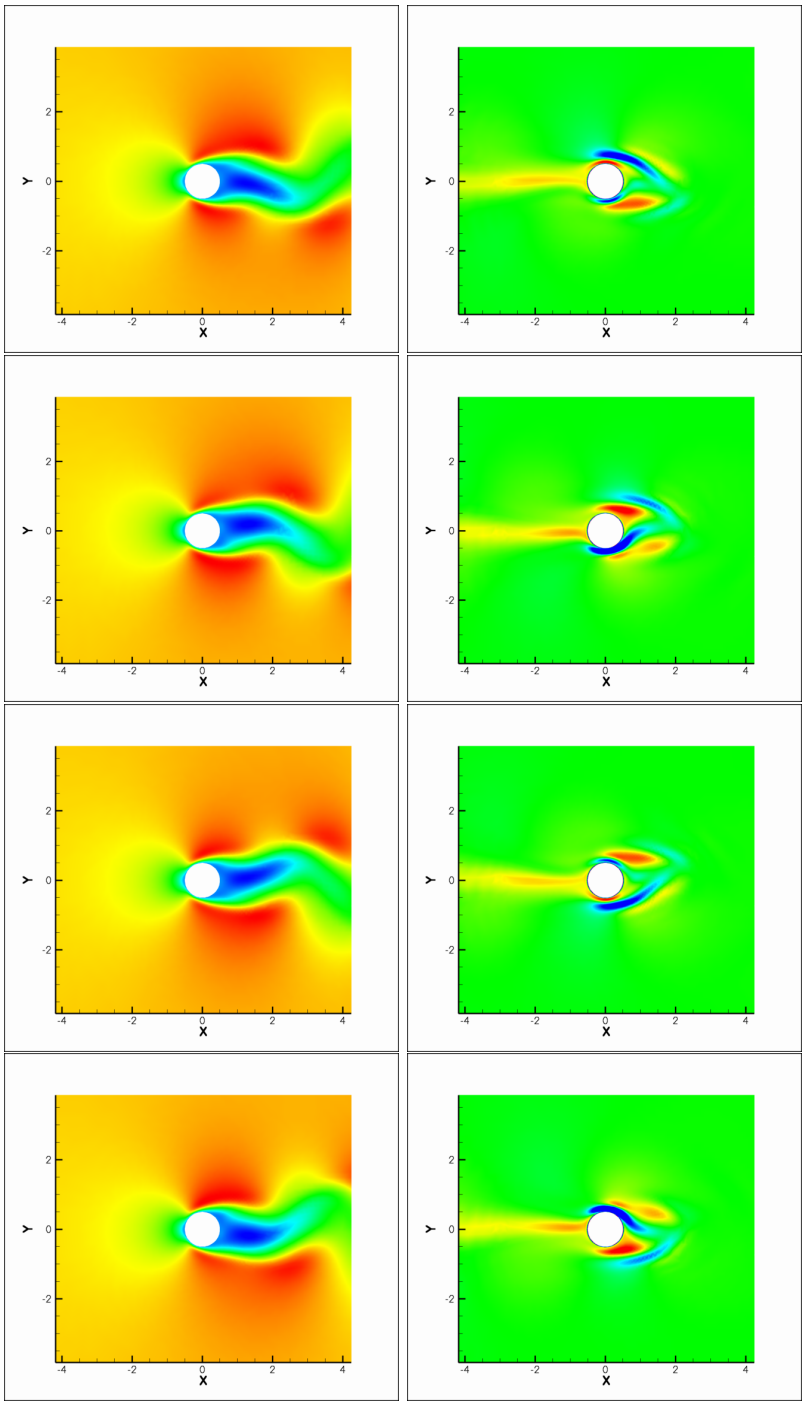

Figure 4.2: Flow and adjoint solutions at $t = 125.0, 126.5, 128.0, 129.5$ (upper-left, upper-right, lower-left, lower-right)

in the shear layers where an increase in u_x would temporarily narrow the wake, decreasing the pressure drag.

By integrating the adjoint solution using (4.26), we obtain the sensitivity gradient of the objective function with respect to the random variables describing the sources of uncertainty. In this case, it is the sensitivity gradient of the time-averaged drag coefficient with respect to the rate of rotation ω. Since ω is a random time process, the sensitivity gradient obtained is a function of time. This function $\hat{\omega} = \left.\frac{\partial \mathbf{J}}{\partial \omega}\right|_t$ is plotted in Figure 4.3, along with the time history of the instantaneous lift and drag coefficients on the cylinder.

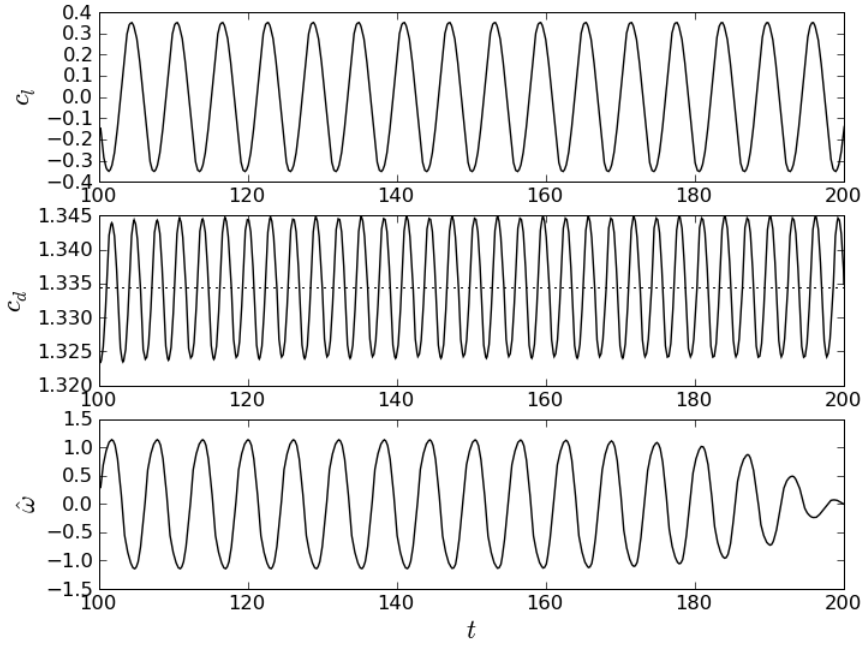

Figure 4.3: The time history of c_l, c_d and $\hat{\omega} = \frac{\partial \mathbf{J}}{\partial \omega}$ of flow past a non-rotating circular cylinder at $Re = 100$. The objective function for the adjoint equation is the time-integrated drag on the cylinder. The time averaged drag coefficient 1.3345 is plotted as a dotted line in the c_d plot.

This example completes our discussion of how to solve the adjoint equation for the unsteady incompressible Navier-Stokes equations, and how to calculate the sensitivity gradient from the adjoint solution. The next two chapters discuss how to use this sensitivity gradient in uncertainty quantification problems.

Chapter 5

Quantification of Margins and Risk using the Adjoint Method

5.1 Introduction

The previous chapters discussed how to solve the adjoint equation for unsteady problems, and how to calculate the sensitivity gradient using the resulting adjoint solution. This chapter and the next present two uncertainty quantification methods that use the sensitivity gradients calculated from the adjoint solution. Discussed in this chapter is a method for failure and risk analysis. These types of analyses require calculating tail probabilities of an objective function. More specifically, we want to calculate the small probability that the objective function exceeds a certain critical value, usually representing the probability that a system fails or suffers catastrophic losses. Mathematically, let $\mathbf{J}(\xi)$ be the objective function, which depends on a vector of uncertain variables ξ. The probability distribution of ξ is known. Let $\mathbf{J_C}$ be a known constant. We want to calculate $P(\mathbf{J} > \mathbf{J_C})$. This probability can be as high as 10%, and as small as 10^{-5} or even smaller.

Calculating such tail probabilities is challenging. Polynomial chaos and collocation based methods do not represent tail probabilities accurately. Monte Carlo is the most commonly used method in this situation. However, the brute force Monte Carlo method can be very computationally expensive and inefficient; if the tail probability is small, only a small fraction of the samples would fall into the tail region, resulting in insufficient sampling. In

fact, let $\xi_i, i = 1, \ldots, N$ be samples of ξ. The brute force Monte Carlo method approximates

$$P(\mathbf{J} > \mathbf{J_C}) \approx P_N = \frac{1}{N} \sum_{i=1}^{N} I(\mathbf{J}(\xi_i) > \mathbf{J_C}) \ ,$$

where $I(\mathbf{J}(\xi_i) > \mathbf{J_C})$ is the indicator function. Its value is 1 if $\mathbf{J}(\xi_i) > \mathbf{J_C}$ is true and zero otherwise. The variance of this estimator P_N is

$$\mathrm{Var}[P_N] = \frac{1}{N} \mathrm{Var}[I(\mathbf{J}(\xi_i) > \mathbf{J_C})] = \frac{P(\mathbf{J} > \mathbf{J_C}) - P(\mathbf{J} > \mathbf{J_C})^2}{N} \ . \tag{5.1}$$

The relative error of the Monte Carlo method can be characterized by the ratio of the standard deviation and the mean of the estimator P_N, which is

$$\frac{\sqrt{\mathrm{Var}[P_N]}}{P(\mathbf{J} > \mathbf{J_C})} = \sqrt{\frac{1 - P(\mathbf{J} > \mathbf{J_C})}{N \, P(\mathbf{J} > \mathbf{J_C})}}.$$

This formula reveals the inefficiency of the brute force Monte Carlo method when the tail probability is small. For a fixed number of samples, the smaller the tail probability $P(\mathbf{J} > \mathbf{J_C})$, the larger the relative error; for a fixed target relative error, the smaller the tail probability, the more samples must be used to meet the target relative error. For example, if the tail probability $P(\mathbf{J} > \mathbf{J_C}) = 5\%$, and the target relative error is 10%, about 2,000 samples should be used. But when the tail probability to be calculated is 0.1%, 100,000 samples should be used. In computationally intensive engineering problems, calculating $\mathbf{J}(\xi_i)$ for each ξ_i is expensive. Therefore, it is impractical to use the brute force Monte Carlo method, and some method of accelerating its convergence rate must be used.

5.2 Accelerating Monte Carlo using adjoint sensitivity gradient

Acceleration of convergence for Monte Carlo methods can be achieved by reducing the variance of its estimator. For a fixed target approximation error, the number of samples required is proportional to the variance of an unbiased estimator. This section discusses variance-reduction techniques for calculating the tail probability $P(\mathbf{J} > \mathbf{J_C})$ based on a single adjoint calculation. The adjoint calculation can be used to calculate the sensitivity gradient, which can be used to approximate the objective function as a linear function of the

random variables describing the sources of uncertainty. Suppose the adjoint is evaluated at ξ_0, and the sensitivity gradient $\mathbf{J}'(\xi_0)$ is calculated from the adjoint solution. The objective function can be approximated in the vicinity of ξ_0 as

$$\mathbf{J}(\xi) \approx \mathbf{J}_L(\xi) = \mathbf{J}(\xi_0) + \nabla\mathbf{J}(\xi_0) \cdot (\xi - \xi_0) \ .$$

The information provided by this linear approximation enables us to reduce the variance of the Monte Carlo method using control variate and importance sampling. Section 5.2.1 discusses the control variate technique, and Section 5.2.2 combines control variate with importance sampling to further reduce the variance.

5.2.1 Control variates

The idea of control variate (Glynn & Szechtman, 2002) explores the similarity between the real objective function $\mathbf{J}$ and its linear approximation $\mathbf{J}_L$. In this method, we approximate the target tail probability $P(\mathbf{J} > \mathbf{J_C})$ with the estimator

$$P(\mathbf{J} > \mathbf{J_C}) \approx P_N^{CV} = P(\mathbf{J}_L > \mathbf{J_C}) + \frac{1}{N}\sum_{i=1}^{N} \left(I(\mathbf{J}(\xi_i) > \mathbf{J_C}) - I(\mathbf{J}_L(\xi_i) > \mathbf{J_C})\right) \ .$$

This estimator P_N^{CV} is unbiased for $P(\mathbf{J} > \mathbf{J_C})$ because

$$\begin{aligned} \mathrm{E}\left[P_N^{CV}\right] &= P(\mathbf{J}_L > \mathbf{J_C}) + \mathrm{E}[I(\mathbf{J}(\xi_i) > \mathbf{J_C})] - \mathrm{E}[I(\mathbf{J}_L(\xi_i) > \mathbf{J_C})] \\ &= P(\mathbf{J}_L > \mathbf{J_C}) + P(\mathbf{J} > \mathbf{J_C}) - P(\mathbf{J}_L > \mathbf{J_C}) \\ &= P(\mathbf{J} > \mathbf{J_C}) \ . \end{aligned}$$

The variance of this new estimator is

$$\begin{aligned} \mathrm{Var}\left[P_N^{CV}\right] =& \frac{1}{N}\,\mathrm{Var}\left[I(\mathbf{J} > \mathbf{J_C}) - I(\mathbf{J}_L > \mathbf{J_C})\right] \\ =& \frac{1}{N}\left(P(\mathbf{J} > \mathbf{J_C} > \mathbf{J}_L \text{ or } \mathbf{J} < \mathbf{J_C} < \mathbf{J}_L) - (P(\mathbf{J}_L > \mathbf{J_C}) - P(\mathbf{J} > \mathbf{J_C}))^2\right) . \end{aligned} \tag{5.2}$$

From this formula, we can see that the variance of the control variate estimator depends on the accuracy the linear approximation $\mathbf{J}_L$. If the approximation is good, then $\mathbf{J} \approx \mathbf{J}_L$. As a result, $P(\mathbf{J} > \mathbf{J_C} > \mathbf{J}_L$ or $\mathbf{J} < \mathbf{J_C} < \mathbf{J}_L)$, which is the probability that $\mathbf{J_C}$ lies within the

small interval $[\mathbf{J}, \mathbf{J}_L]$, is small. Since $\text{Var}\left[P_N^{CV}\right] \leq \frac{1}{N} P(\mathbf{J} > \mathbf{J_C} > \mathbf{J}_L \text{ or } \mathbf{J} < \mathbf{J_C} < \mathbf{J}_L)$, the variance is small. In fact, if the approximation is exact, i.e., $\mathbf{J}_L = \mathbf{J}$, then the variance is zero. However, if the approximation is not accurate at all, this estimator may not reduce the variance. For example, if the $\mathbf{J}_L < \mathbf{J_C}$ is a constant, then $P(\mathbf{J} > \mathbf{J_C} > \mathbf{J}_L \text{ or } \mathbf{J} < \mathbf{J_C} < \mathbf{J}_L) = P(\mathbf{J} > \mathbf{J_C})$, and $\text{Var}\left[P_N^{CV}\right] = \frac{1}{N}\left(P(\mathbf{J} > \mathbf{J_C}) - P(\mathbf{J}_L > \mathbf{J_C})^2\right)$ is the same as the variance of the Naive Monte Carlo method. For this reason, this method offers the most improvement in convergence when the linear approximation $\mathbf{J}_L$ is at least a fairly accurate approximation.

In Section 5.3, we will apply the control variate technique in an unsteady fluid flow problem, and demonstrate its variance-reduction effectiveness.

5.2.2 Importance sampling

The brute force Monte Carlo method is ineffective when the target failure probability is small because only a small fraction of samples lay in the region where failure might occur. The importance sampling technique (Glynn & Iglehart, 1989) addresses this ineffectiveness by concentrating the samples in regions where the failure is more likely to occur.

In our adjoint-based importance sampling, we select the concentration of sampling based on how likely the objective function $\mathbf{J}$ and its linear approximation $\mathbf{J}_L$ are on different sides of the critical value $\mathbf{J_C}$. Therefore, the first step of our method is to estimate the error of this linear approximation; since it is likely to grow as $O(|\xi - \xi_0|^2)$, we model the normalized approximation error $\dfrac{\mathbf{J} - \mathbf{J}_L}{||\xi - \xi_0||^2}$ as a random variable, whose probability distribution is estimated by plotting the histogram of $\dfrac{\mathbf{J}(\xi_i) - \mathbf{J}_L(\xi_i)}{||\xi_i - \xi_0||^2}$ for a small number of samples ξ_i. This step quantifies how much the objective function $\mathbf{J}$ may deviate from its linear approximation $\mathbf{J}_L$. For each ξ, without going through the expensive process of calculating the objective function $\mathbf{J}(\xi)$, we can estimate its prior probability distribution based only on its linear approximation $\mathbf{J}_L(\xi)$ and $|\xi - \xi_0|^2$. Note that this probability distribution is defined in the Bayesian sense, in contrast to the tail probability we intend to calculate, which is defined as a frequency probability. For the sake of clarity, we denote the Bayesian probability as p and the frequency probability as P.

Now for each ξ, we can calculate the prior probability

$$p(\mathbf{J}(\xi) > \mathbf{J_C} \mid \mathbf{J}_L(\xi)) = p\left(\frac{\mathbf{J}(\xi) - \mathbf{J}_L(\xi)}{||\xi - \xi_0||^2} > \frac{\mathbf{J_C} - \mathbf{J}_L(\xi)}{||\xi - \xi_0||^2} \;\middle|\; \mathbf{J}_L(\xi)\right)$$

based on the value of $\mathbf{J}_L(\xi)$ and the estimated probability distribution of $\frac{\mathbf{J}(\xi)-\mathbf{J}_L(\xi)}{||\xi-\xi_0||^2}$. We call it *a prior* probability because this probability is estimated prior to calculating the real value of the objective function $\mathbf{J}(\xi)$. Furthermore, we can calculate the prior probability that the objective function and its linear approximation are on different sides of $\mathbf{J_C}$. We denote this probability as p_D,

$$\begin{aligned} p_D(\xi) &= p(\mathbf{J}(\xi) > \mathbf{J_C} > \mathbf{J}_L(\xi) \text{ or } \mathbf{J}(\xi) < \mathbf{J_C} < \mathbf{J}_L(\xi) \mid \mathbf{J}_L(\xi)) \\ &= \begin{cases} p(\mathbf{J}(\xi) > \mathbf{J_C} \mid \mathbf{J}_L(\xi)), & \mathbf{J_C} > \mathbf{J}_L(\xi) \\ 1 - p(\mathbf{J}(\xi) > \mathbf{J_C} \mid \mathbf{J}_L(\xi)), & \mathbf{J_C} < \mathbf{J}_L(\xi)\ . \end{cases} \end{aligned}$$

Our importance sampling technique concentrates the samples based on this prior probability. For each proposed sample ξ_i obtained according to the distribution of ξ, we calculate $\mathbf{J}_L(\xi_i)$ and $p_D(\xi)$. This probability determines the rate at which the proposed sample is approved or rejected. Specifically, an independent uniform pseudo-random number is generated and compared with this prior probability, and the sample is rejected if the pseudo-random number is greater of the two. Calculations for $\mathbf{J}(\xi_i)$ are done for approved ξ_is only. This mechanism concentrates the approved samples in regions where the linear approximation $\mathbf{J}_L$ is likely to give a false indication of whether the objective function exceeds the critical value $\mathbf{J_C}$.

In order to achieve an unbiased estimator, this concentration of samples must be compensated for by weighing each sample differently. Let $f(\xi)$ be the probability density function of ξ, then the probability density function of the approved samples is $\frac{f(\xi)p_D(\xi)}{\int p_D(\xi)dP(\xi)}$; if we denote this probability distribution as $Q(\xi)$, then the Radon-Nikodym derivative is given by

$$\frac{dQ}{dP} = \frac{p_D(\xi)}{\int p_D(\xi)dP(\xi)}\ . \tag{5.3}$$

With the approved samples $\xi_1, \xi_2, \ldots, \xi_N$ distributed according to the new probability distribution Q, a different estimator P_N^{IS} is used.

$$P(\mathbf{J}(\xi) > \mathbf{J_C}) \approx P_N^{IS} = P(\mathbf{J}_L > \mathbf{J_C}) + \frac{1}{N}\sum_{i=1}^{N} \frac{dP}{dQ}\ (I(\mathbf{J}(\xi_i) > \mathbf{J_C}) - I(\mathbf{J}_L(\xi_i) > \mathbf{J_C}))\,.$$

This estimator P_N^{IS} is unbiased for $P(\mathbf{J} > \mathbf{J_C})$ because

$$\begin{aligned}
\mathrm{E}_Q\left[P_N^{IS}\right] &= P(\mathbf{J}_L > \mathbf{J_C}) + \mathrm{E}_Q\left[\frac{dP}{dQ}\left(I(\mathbf{J}(\xi_i) > \mathbf{J_C}) - I(\mathbf{J}_L(\xi_i) > \mathbf{J_C})\right)\right] \\
&= P(\mathbf{J}_L > \mathbf{J_C}) + \mathrm{E}_P\left[I(\mathbf{J}(\xi_i) > \mathbf{J_C}) - I(\mathbf{J}_L(\xi_i) > \mathbf{J_C})\right] \\
&= P(\mathbf{J} > \mathbf{J_C})\,.
\end{aligned}$$

The new estimator P_N^{IS} with importance sampling can further reduce the variance of the Monte Carlo method and accelerate its convergence. The variance of this importance sampling estimator is

$$\begin{aligned}
\mathrm{Var}_Q\left[P_N^{IS}\right] =& \frac{1}{N}\,\mathrm{Var}_Q\left[\frac{dP}{dQ}\left(I(\mathbf{J}(\xi_i) > \mathbf{J_C}) - I(\mathbf{J}_L(\xi_i) > \mathbf{J_C})\right)\right] \\
=& \frac{1}{N}\,\mathrm{E}_Q\left[\left(\frac{dP}{dQ}\right)^2\left(I(\mathbf{J}(\xi_i) > \mathbf{J_C}) - I(\mathbf{J}_L(\xi_i) > \mathbf{J_C})\right)^2\right] \\
& - \frac{1}{N}\left(P(\mathbf{J} > \mathbf{J_C}) - P(\mathbf{J}_L > \mathbf{J_C})\right)^2 \\
=& \frac{1}{N}\,\mathrm{E}_P\left[\frac{dP}{dQ}\,I(\mathbf{J} > \mathbf{J_C} > \mathbf{J}_L \text{ or } \mathbf{J} < \mathbf{J_C} < \mathbf{J}_L)\right] \\
& - \frac{1}{N}\left(P(\mathbf{J} > \mathbf{J_C}) - P(\mathbf{J}_L > \mathbf{J_C})\right)^2 \\
=& \frac{1}{N}\,\mathrm{E}_P\left[\frac{I(\mathbf{J} > \mathbf{J_C} > \mathbf{J}_L \text{ or } \mathbf{J} < \mathbf{J_C} < \mathbf{J}_L)}{p(\mathbf{J} > \mathbf{J_C} > \mathbf{J}_L \text{ or } \mathbf{J} < \mathbf{J_C} < \mathbf{J}_L)}\right]\int p_D(\xi)dP(\xi) \\
& - \frac{1}{N}\left(P(\mathbf{J} > \mathbf{J_C}) - P(\mathbf{J}_L > \mathbf{J_C})\right)^2\,.
\end{aligned} \tag{5.4}$$

In the next section, we will demonstrate that the importance sampling technique indeed reduces the variance.

5.3 Application to an unsteady fluid flow problem

We consider the laminar flow around a circular cylinder at a Reynolds number of 100. We use the same mesh, problem setup and numerical methods as described in Section 4.3. The objective function is the time-averaged drag coefficient of the cylinder, and we want to calculate the probability that it is greater than a critical value $\mathbf{J_C} = 1.345$.

The uncertainties in the problem come from small, unknown rotational oscillations of the cylinder in the flow. We assume that this random rotation consists of 10 different

frequencies, including the vortex-shedding frequency and its subharmonics. The rotation at each frequency is described by two random numbers, making each frequency component of the oscillation random both in magnitude and random in phase. Specifically, let f_V be the frequency of the vortex-shedding, the angular speed of the cylinder rotating about its symmetric axis is

$$w(t) = \sum_{i=1}^{10} \xi_{2i-1} \cos(2\pi i f_V t) + \xi_{2i} \sin(2\pi i f_V t) \,, \tag{5.5}$$

where $\xi_1, \ldots, \xi_{20}$ are Gaussian random variables with mean 0 and standard deviation 0.01.

Although the magnitude of the random rotational oscillations are small, they have significant impact on the flowfield. As a result, the forces exerted on the cylinder by the fluid can be significantly affected. Figure 5.1 plots the time history of the drag coefficient c_d and lift coefficient c_l of the cylinder for 1000 samples of the random oscillations. These non-dimensionalized quantities c_d and c_l are defined as

$$c_d = \frac{D}{\frac{1}{2}\rho v_\infty^2 D\, b} \qquad c_l = \frac{L}{\frac{1}{2}\rho v_\infty^2 D\, b}$$

where D is the drag of the cylinder, i.e., the component of the aerodynamic force that is parallel to the freestream; L is the lift of the cylinder, i.e the component of the aerodynamic force that is perpendicular to the freestream; ρ is the density of the fluid; v_∞ is the speed of the freestream flow; D is the diameter of the cylinder, and b is the spanwise length of the cylinder.

In Figure 5.1, the white lines represent the lift and drag coefficients with no random oscillations. The black dotted lines show the lift and drag coefficients of cylinders undergoing rotational oscillations described by equation (5.5) with 1000 samples of the random vector $(\xi_1, \ldots, \xi_{20})$. As can be seen, the drag coefficient can be significantly changed by the random rotational oscillations. Therefore, our objective function, the time-averaged drag coefficient $\mathbf{J}$, should also depend on the specific form of the rotational oscillation, which is specified by the random vector $(\xi_1, \ldots, \xi_{20})$.

Figure 5.2 shows that the objective function is indeed modified by the random rotational oscillations. This histogram shows the empirical density function for the time-averaged drag coefficient, our objective function $\mathbf{J}$. The dotted vertical line represents the critical value $\mathbf{J_C} = 1.345$. As can be seen, there is a small but non-trivial probability that our objective function is higher than this critical value. We can also roughly estimate this tail probability

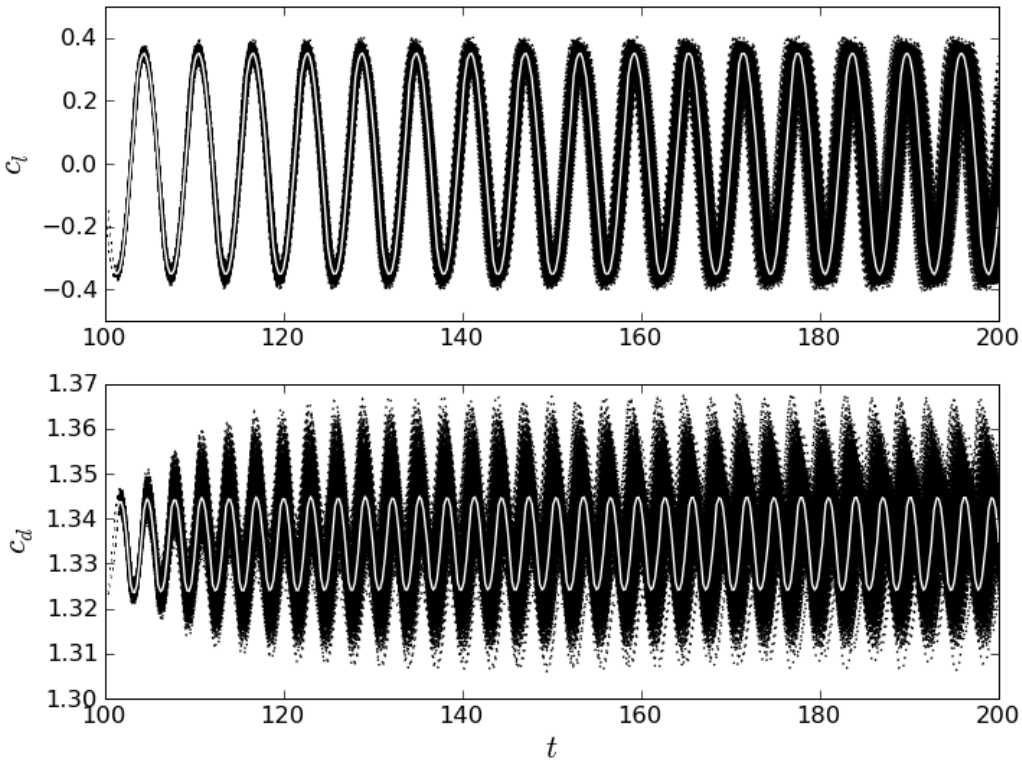

Figure 5.1: Realizations of the c_l (the top figure) and c_d (the bottom figure). The white lines indicate c_l and c_d with no rotations.

from the histogram. Since the objective function exceeds the critical value in 48 of the 1000 samples we calculated, an unbiased estimate of the tail probability is $P(\mathbf{J} > \mathbf{J_C}) \approx 4.8\%$. However, the standard deviation of this estimate is 0.007, making the 3σ confidence interval of the tail probability

$$P(\mathbf{J} > \mathbf{J_C}) = 4.8 \pm 2.0\%. \tag{5.6}$$

These numbers reveal the inefficiency of the brute force Monte Carlo method. Despite of calculating the objective function for 1000 samples, the fraction of the samples whose objective function is higher than the critical value is small. As a result, the probability calculated by this method has a large variance and therefore is not accurate.

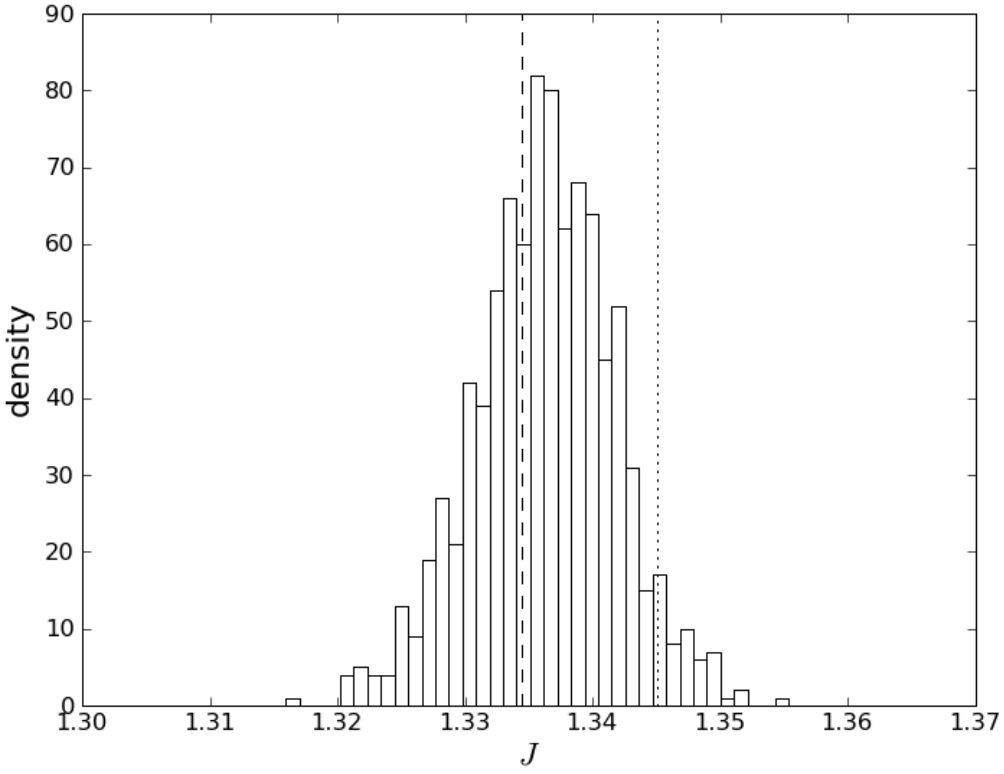

Figure 5.2: The histogram of the objective function $\mathbf{J} = \overline{c_d}$. The dashed vertical line indicates $\mathbf{J}$ with no rotation. The dotted vertical line indicates $\mathbf{J_C} = 1.345$.

5.3.1 Adjoint linear approximation

The adjoint equation for this problem is solved with the method described in Chapters 3 and 4. From the adjoint solution, we can calculate the gradient of the objective function with respect to the rate of rotation of the cylinder.

$$\hat{\omega}(t) = \frac{\partial \mathbf{J}}{\partial \omega(t)}$$

Figure 4.3 (previous chapter) shows the time history of $\hat{\omega}$. With $\hat{\omega}$ calculated, we can obtain the sensitivity gradient of the objective function with respect to $\xi_1, \ldots, \xi_{20}$, the

random variables describing the rotational oscillation.

$$\begin{aligned}\frac{\partial \mathbf{J}}{\partial \xi_i} &= \int_{T_0}^{T_1} \frac{\partial \mathbf{J}}{\partial \omega(t)} \frac{\partial \omega(t)}{\partial \xi_i}\, dt \\ &= \begin{cases} \displaystyle\int_{T_0}^{T_1} \hat{\omega} \cos\left(2\pi \frac{i+1}{2} f_V t\right) dt, & i \text{ is odd} \\ \displaystyle\int_{T_0}^{T_1} \hat{\omega} \sin\left(2\pi \frac{i}{2} f_V t\right) dt, & i \text{ is even} \end{cases}\end{aligned}$$

Calculating this sensitivity derivatives for each $i = 1, \ldots, 20$ generates the sensitivity gradient vector of $\mathbf{J}$ as a function of the random variables $\xi = (\xi_1, \ldots, \xi_{20})$.

$$\nabla \mathbf{J} = \left(\frac{\partial \mathbf{J}}{\partial \xi_1}, \ldots, \frac{\partial \mathbf{J}}{\partial \xi_{20}}\right)$$

With this sensitivity gradient, we can construct a linear approximation of the objective function

$$\mathbf{J}(\xi) \approx \mathbf{J}_L(\xi) = \mathbf{J}_0 + \nabla \mathbf{J} \cdot \xi \tag{5.7}$$

where $\mathbf{J}_0 = \mathbf{J}(0)$ is the value of the objective function when $\xi = 0$, i.e., when the cylinder is not rotating. This adjoint linear approximation can be obtained with one Navier-Stokes solution, from which $\mathbf{J}_0$ is calculated, and one adjoint solution, from which the sensitivity gradient $\nabla \mathbf{J}$ is calculated. Once we have $\mathbf{J}_0$ and $\nabla \mathbf{J}$, the adjoint linear approximation $\mathbf{J}_L(\xi)$ can be calculated using (5.7) at essentially no additional computational cost.

Figure 5.3 shows the true value of the objective function $\mathbf{J}(\xi_i)$ against its adjoint linear approximation $\mathbf{J}_L(\xi_i)$ for 1000 randomly sampled $\xi_1, \ldots, \xi_{1000}$. Each cross on the plot represents one sample. As can be seen, although some samples deviate from the diagonal line, indicating large approximation errors, the adjoint linear approximation is a sufficiently accurate approximation to the objective function.

The adjoint linear approximation can be directly used to obtain a very efficient first-order estimate of $P(\mathbf{J} > \mathbf{J_C})$, the probability we want to calculate. Because $\mathbf{J}_L(\xi)$ can be evaluated with essential no computational cost, a very large number of Monte Carlo samples can be used to accurately calculate $P(\mathbf{J}_L > \mathbf{J_C})$. This probability that the linear approximation exceeds the critical value can be used to approximate the probability that the objective function exceeds the critical value. $P(\mathbf{J} > \mathbf{J_C}) \approx P(\mathbf{J}_L > \mathbf{J_C})$.

Figure 5.4 shows the application of this approach to our cylinder problem. The solid line

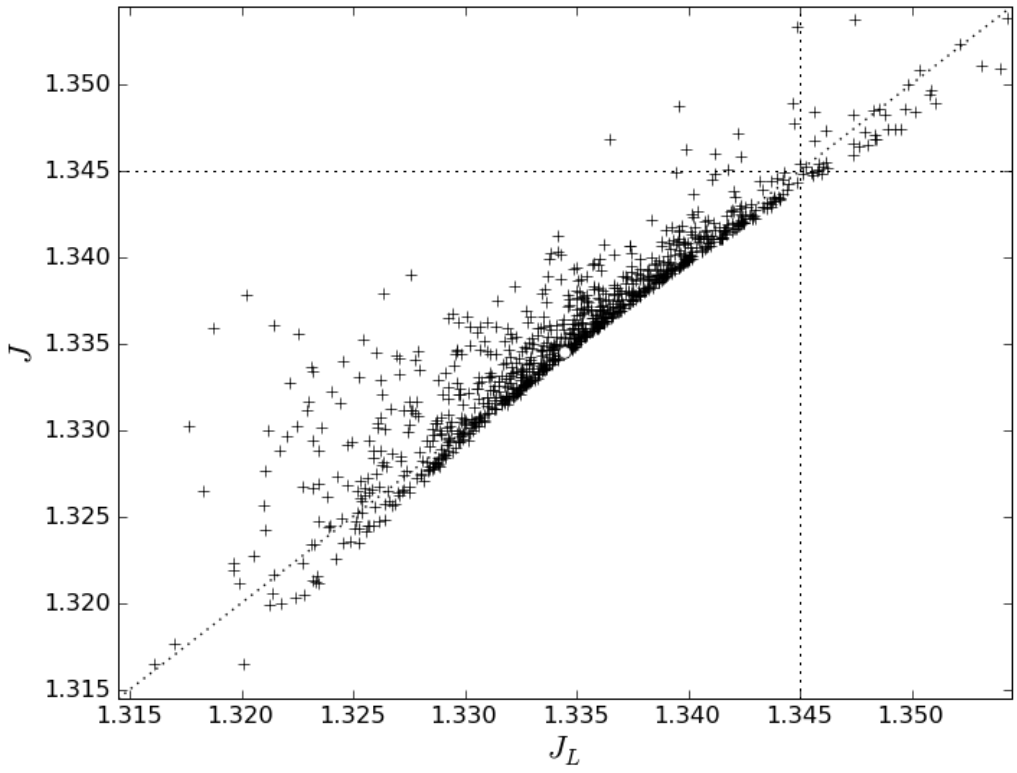

Figure 5.3: The correlation between adjoint approximation (horizontal axis) and true objective function (vertical axis). The horizontal and vertical dotted lines indicates the critical value $\mathbf{J_C}$. The circular symbol at the center indicates the objective function without rotation.

is the empirical distribution function obtained from 10,000,000 samples of the adjoint linear approximation $\mathbf{J}_L$. For comparison, the bars are the same empirical distribution function obtained from the 1000 samples of $\mathbf{J}$. The vertical dotted line indicates the critical value $\mathbf{J_C}$. As can be seen, the distribution function of the adjoint linear approximation is close to the distribution of the true objective function. In addition, because the distribution of the adjoint approximation is obtained with many more samples, it is much smoother than the empirical distribution of the true objective function. At the same time, the tail probability for the adjoint approximation $P(\mathbf{J}_L > \mathbf{J_C})$ can be obtained with little variance. In this case, we calculated

$$P(\mathbf{J} > \mathbf{J_C}) \approx P(\mathbf{J}_L > \mathbf{J_C}) = 3.6\%. \tag{5.8}$$

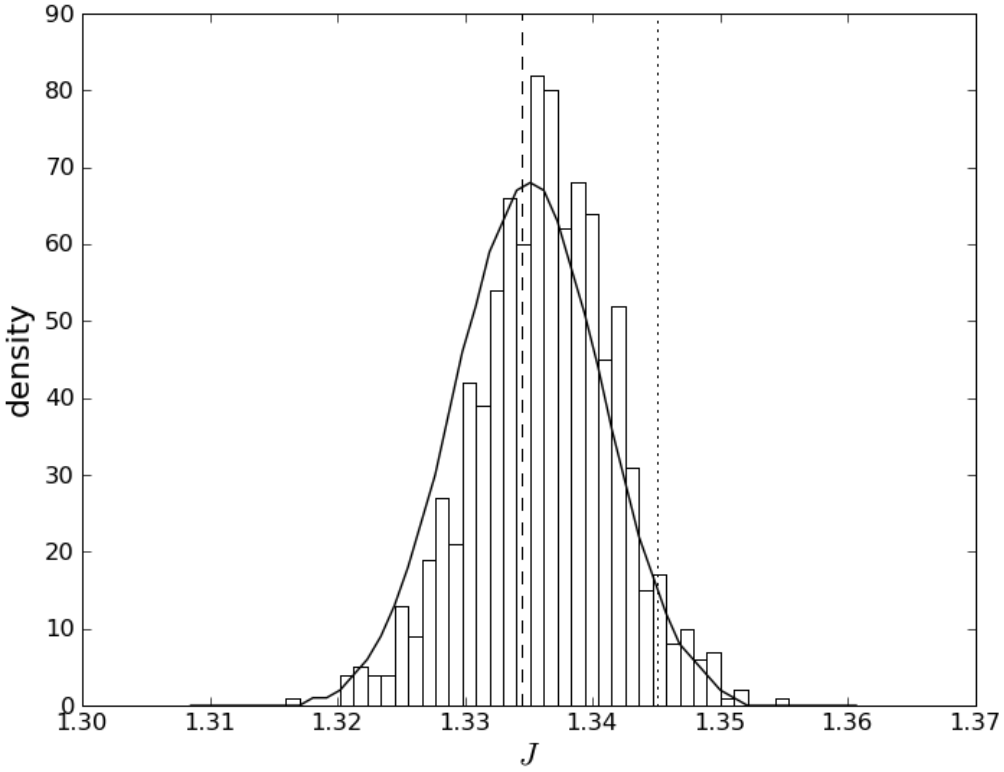

Figure 5.4: The empirical density function of the objective function obtained using the brute-force Monte Carlo (vertical bars), and the empirical density function of the adjoint approximation obtained using adjoint method (solid line). The dashed vertical line indicates $\mathbf{J}$ with no rotation. The dotted vertical line indicates $\mathbf{J_C} = 1.345$.

We compare this result with the value calculated using the brute force Monte Carlo method (5.6). The accuracy of their results are similar, because the true value of $P(\mathbf{J} > \mathbf{J_C})$ is $4.1 \pm 0.5\%$ (as calculated in Equation (5.10) later in this chapter). Nevertheless, the error of this adjoint approximation method is different in nature from the error of the brute force Monte Carlo method. The error of the brute force Monte Carlo method comes from its large variance of the estimator, while the result of the adjoint approximation method has very little variance due to the large number of samples used. Instead, the error of the adjoint approximation method is a consequence of approximating $P(\mathbf{J} > \mathbf{J_C})$ with $P(\mathbf{J}_L > \mathbf{J_C})$. The difference between these two probabilities can be illustrated using Figure 5.3. In the plot, the vertical axis is $\mathbf{J}$, and the horizontal axis is $\mathbf{J}_L$. The probability $P(\mathbf{J} > \mathbf{J_C})$ we

want to calculate is the proportion of samples above the horizontal dotted line. In contrast, $P(\mathbf{J}_L > \mathbf{J_C})$, the probability we use to approximate $P(\mathbf{J} > \mathbf{J_C})$, is the proportion of samples to the right of the vertical line. By using this approximation, we underestimate the samples to the upper-left of the intersection of the two dotted lines, and overestimate the samples to the lower-right of the intersection.

Although the adjoint approximation method has a similar large error to the brute force Monte Carlo method in our example, it is by far less expensive. The brute force Monte Carlo method requires 1000 Navier-Stokes calculations, while the adjoint approximation method involves only one Navier-Stokes calculation and one adjoint calculation. Since the adjoint calculation requires about 4 times the computational resources of a Navier-Stokes calculation, the brute force Monte Carlo method is 200 times more expensive than the adjoint approximation method. This extreme efficiency makes the adjoint method the first choice for a rough estimate of the tail probability.

5.3.2 Accelerated Monte Carlo

Although the adjoint approximation method of estimating $P(\mathbf{J} > \mathbf{J_C})$ is computationally economical, it does not produce very accurate results. In this section, we use the methods described in the previous section to reduce the variance of the Monte Carlo method. With these variance-reduction techniques, we produce more accurate results with the same cost as the brute force Monte Carlo method.

We first apply the control variates technique to this problem, as described in Section 5.2.1. With this technique, we use the same samples as in the brute force Monte Carlo method, but change the estimator to

$$P_N^{CV} = P(\mathbf{J}_L > \mathbf{J_C}) + \frac{1}{N}\sum_{i=1}^{N} \left(I(\mathbf{J}(\xi_i) > \mathbf{J_C}) - I(\mathbf{J}_L(\xi_i) > \mathbf{J_C}) \right).$$

We note that the first term in the estimator is simply the result we obtain from the adjoint approximation method, and the second term is simply an unbiased estimator of the difference between the true tail probability $P(\mathbf{J} > \mathbf{J_C})$ and the estimated tail probability by adjoint approximation $P(\mathbf{J}_L > \mathbf{J_C})$. In Figure 5.3, this estimator is counting the proportion of samples to the upper-left of the intersection, minus the proportion of samples to the lower-right of the intersection. With this estimator and the same 1000 samples as the

brute force Monte Carlo method, we calculate

$$P(\mathbf{J} > \mathbf{J_C}) = 4.1 \pm 1.2\%, \tag{5.9}$$

where the 3σ confidence interval is obtained from the variance formula (5.2). Comparing to (5.6), the variance of the Monte Carlo method is significantly reduced. To achieve the same variance-reduction using the brute force Monte Carlo method, 1778 more Navier-Stokes equations must be solved. In contrast, we achieved the same reduction in variance with merely one additional Navier-Stokes solution and one adjoint solution. This implies that one single adjoint solution, whose cost is 4 times that of a Navier-Stokes solution, achieves the same effect as 1,777 Navier-Stokes equations in this example.

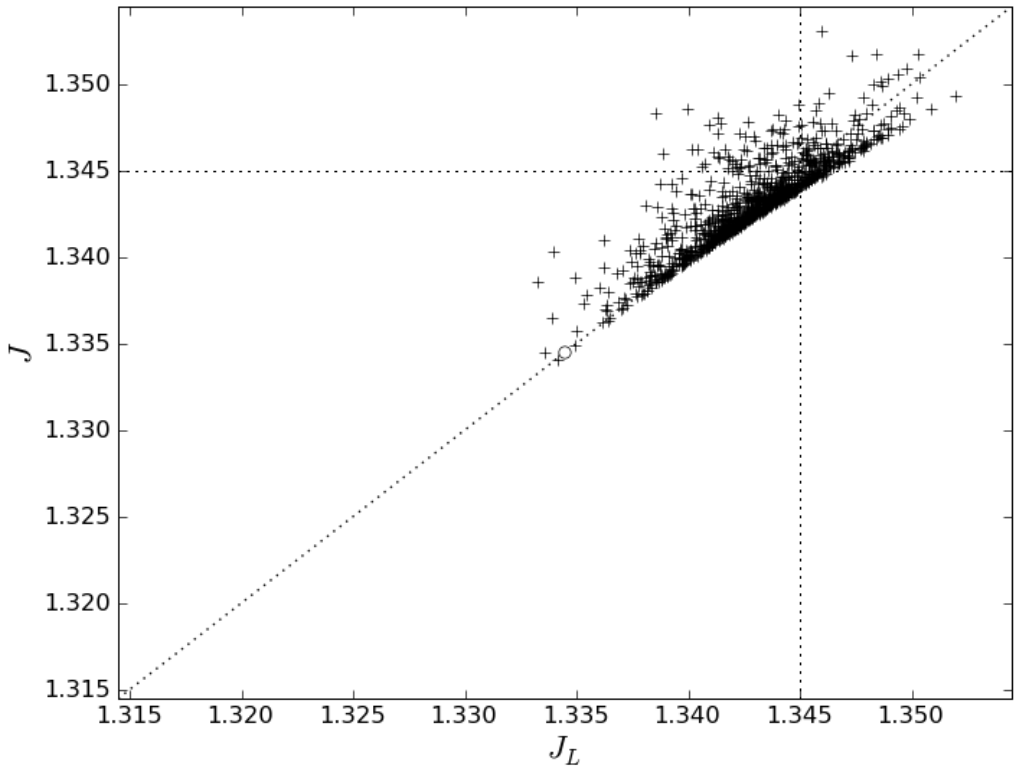

Figure 5.5: Monte Carlo samples with importance sampling. The horizontal axis is the adjoint approximation; the vertical axis is true objective function. The circular symbol at the center indicates the objective function without rotation.

The variance can be further reduced using the importance sampling technique as discussed in Section 5.2.2. We use 25 samples to calculate the variance of the normalized approximation error $\frac{\mathbf{J}-\mathbf{J}_L}{||\xi-\xi_0||^2}$, and approximate it with a Gaussian distribution with zero mean and same variance. This Gaussian distribution is used to calculate $p(\mathbf{J}(\xi) > \mathbf{J_C})$ for each ξ, the probability that a proposed sample is approved. The resulting samples distribute according to probability Q (5.3), and concentrate in areas where the adjoint approximation $P(\mathbf{J}_L > \mathbf{J_C}) \approx P(\mathbf{J} > \mathbf{J_C})$ is likely to be erroneous. This includes the regions where $\mathbf{J}$ is close to $\mathbf{J_C}$, and areas where $\mathbf{J}$ may be significantly different from its adjoint approximation $\mathbf{J}_L$.

Figure 5.5 shows the concentration of the approved samples used in importance sampling. We plot the true value of the objective function $\mathbf{J}(\xi_i)$ against its adjoint linear approximation $\mathbf{J}_L(\xi_i)$ for $\xi_1, \ldots, \xi_{1000}$ sampled from probability distribution Q. Each cross on the plot represents one sample. The diagonal dotted line indicates where the horizontal axis is equal to the vertical axis; the horizontal and vertical dotted lines indicate the critical value $\mathbf{J_C}$. As can be seen, most of the samples concentrate near this critical value. Compared to Figure 5.3, there are many more samples to the upper-left and to the lower-right of the intersection. These additional samples allows a much-reduced variance in estimating the errors made by the adjoint approximation $P(\mathbf{J}_L > \mathbf{J_C})$, and returns a significantly more accurate results. With this method, we calculated

$$P(\mathbf{J} > \mathbf{J_C}) = 4.1 \pm 0.5\%, \tag{5.10}$$

where the 3σ confidence interval is based on the variance formula (5.4). Comparing to the result obtained by the brute force Monte Carlo method (5.6), the variance of the Monte Carlo method is significantly reduced by 75%. To achieve the same variance-reduction using the brute force Monte Carlo method, 15,000 more Navier-Stokes equations would have to be solved. In contrast, we achieved the same reduction in variance with 26 additional Navier-Stokes solutions and one adjoint solution. This implies that one single adjoint solution, whose cost is 4 times that of a Navier-Stokes solution, achieves the same effect as 14974 Navier-Stokes equations in this example.

Figure 5.6 compares the convergence of three different Monte Carlo methods. Different colors represent different methods: red is brute-force Monte Carlo, blue is with redesigned estimator but without importance sampling, and black is with importance sampling. It can

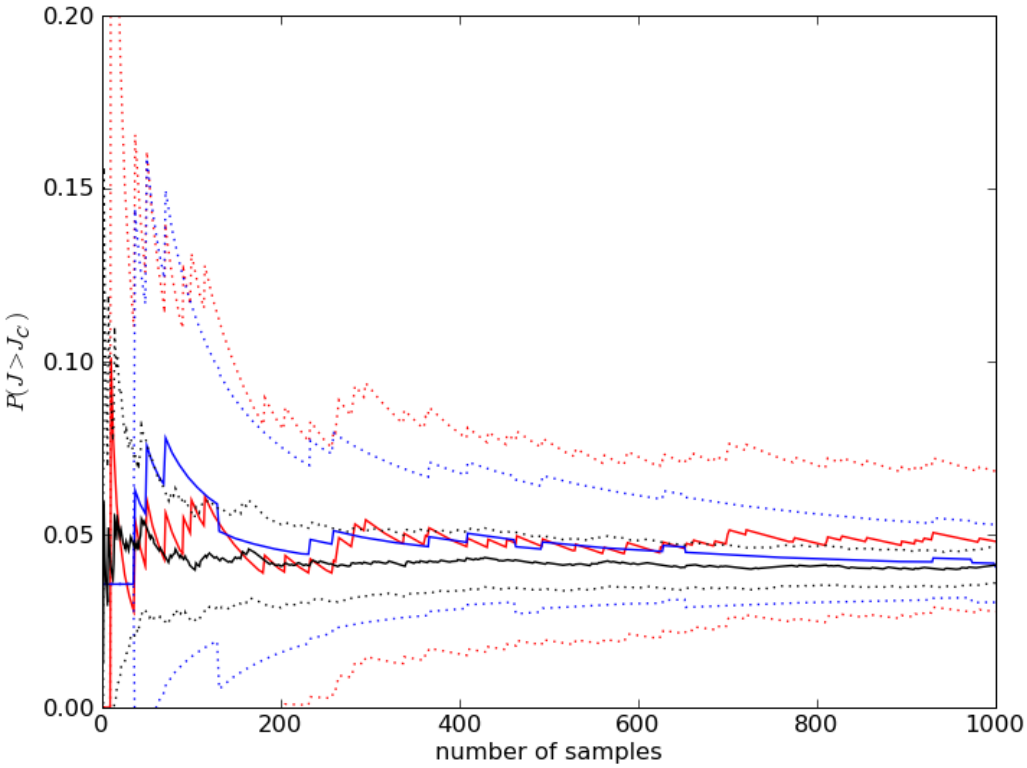

Figure 5.6: Convergence history of three different Monte Carlo methods: Red is brute-force Monte Carlo method; blue is Monte Carlo with control variate; black is Monte Carlo method with control variates and importance sampling. The horizontal axis indicates the number of samples; the solid lines are the $P(\mathbf{J} > \mathbf{J_C})$ calculated by the estimators of each method; the dotted lines are the 3σ confidence interval bounds of the estimators.

be seen that using the better estimator with importance sampling reduces the standard deviation by a factor of 4, which implies that the number of samples required is reduced by a factor of 16.

5.4 Conclusion

Table 5.1 compares the computational cost, accuracy and efficiency of the four methods discussed in this chapter. As can be seen, using the adjoint equation can significantly increase the accuracy and reduce the computational time in quantification of margins and

Table 5.1: Comparison of the methods for estimating $P(\mathbf{J} > \mathbf{J_C})$.

	Computational time	3σ	Equivalent samples
Brute force Monte Carlo	240 hours	2.0%	1,000
Adjoint approximation	1 hour	—	—
Control variate	241 hours	1.2%	2,778
Importance sampling	247 hours	0.5%	16,000

risk. The adjoint approximation method is by far less expensive than any Monte Carlo method, and is often sufficiently accurate for a first estimate. However, it is difficult to evaluate the accuracy of its result. Both the control variate method and the importance sampling method are accelerated Monte Carlo with adjoint solution. They are marginally more expensive than the brute force Monte Carlo method, but the accuracy is significantly improved.

Chapter 6

Gradient-based Interpolation using the Adjoint Method

6.1 Introduction

This chapter further explores efficient use of adjoint solutions in uncertainty quantification. When many random variables are used to describe the uncertainty sources, an adjoint solution reveals much more information about the quantities of interest than a solution of the primal equation, such as the Navier-Stokes equations. In the previous chapter, we have demonstrated that thousands of Navier-Stokes solutions can be replaced by a single adjoint solution. In this chapter, we aim to further improve the efficiency of uncertainty propagation methods by employing more than one adjoint solution.

When multiple adjoint equations are solved for different samples of the random variables describing uncertainty sources, we obtain the sensitivity gradient of the objective function at different locations in the parameter space. We can use this gradient information, together with the function values at these points, to construct an interpolation approximation of the objective function. Since evaluating the interpolant is much cheaper than solving the primal problem, we can calculate statistical information and probabilities simply using the Monte Carlo method and sampling from the interpolant. If the interpolation scheme is accurate, the statistics and probabilities of the interpolant are accurate representations of that of the true objective function.

This chapter presents a novel numerical scheme for constructing an accurate interpolant using adjoint sensitivity gradients. We represent each datapoint using a Taylor series and

assume the derivatives in the series to be random variables. The interpolation weights of the datapoints are then chosen to minimize the expectation of the interpolation error. Using this formulation, the gradient information on each datapoint can be used to significantly reduce the interpolation error. We show that our interpolation converges exponentially on smooth functions for a variety of grids, including randomly scattered data points.

6.2 Interpolation in one-dimensional space

6.2.1 Mathematical formulation

Our interpolation scheme approximates the value of a function f using an interpolant function $\hat{f}$ calculated from $n+m$ datapoints. Let $\hat{f}(x_i)$ be the measurements of the function values at n given "value nodes" $x_i, i = 1, \ldots, n$ (their corresponding datapoints are denoted as "value datapoints"), and $\hat{f}'(y_i)$ the measurements of the function derivatives at m "gradient nodes" $y_i, i = 1, \ldots, m$ (their corresponding datapoints are denoted as "gradient datapoints"). The value nodes x_i and gradient nodes y_i are allowed to be (and often are) the same. The measurement errors are assumed to be mutually independent random variables with zero mean, and their variances are given as

$$e_{a\,i}^2 = E\left[(\hat{f}(x_i) - f(x_i))^2\right], \quad i = 1, \ldots, n,$$

$$e_{b\,i}^2 = E\left[(\hat{f}'(y_i) - f'(y_i))^2\right], \quad i = 1, \ldots, m,$$

which can be set to 0 for exact measurements of the function. The value of our interpolant at point z is a linear combination of the measurements $\hat{f}(x_i)$ and $\hat{f}'(y_i)$:

$$\tilde{f}(z) = \sum_{i=1}^{n} a_i \hat{f}(x_i) + \sum_{i=1}^{m} b_i \hat{f}'(y_i), \tag{6.1}$$

where the coefficients a_i satisfy the normalization condition

$$\sum_{i=1}^{n} a_i = 1. \tag{6.2}$$

Each $f(x_i)$ can be expanded using Taylor's theorem:

$$f(x_i) = f(z) + \sum_{k=1}^{N} f^{(k)}(z)\frac{(x_i - z)^k}{k!} + f^{(N+1)}(\xi_i)\frac{(x_i - z)^{N+1}}{(N+1)!},$$

where each ξ_i is between x_i and z. We also expand each $f'(y_i)$:

$$f'(y_i) = \sum_{k=1}^{N} f^{(k)}(z)\frac{(y_i - z)^{k-1}}{(k-1)!} + f^{(N+1)}(\eta_i)\frac{(y_i - z)^{N}}{N!},$$

where each η_i is between y_i and z. The total order of the expansion is set to

$$N = n + m$$

in this report. Substituting these expansions into (6.1), the residual of the interpolation is

$$\begin{aligned} r(z) = \tilde{f}(z) - f(z) = &\sum_{k=1}^{N} f^{(k)}(z)\left(\sum_{i=1}^{n} a_i\frac{(x_i - z)^k}{k!} + \sum_{i=1}^{m} b_i\frac{(y_i - z)^{k-1}}{(k-1)!}\right) \\ &+ \sum_{i=1}^{n} f^{(N+1)}(\xi_i)\left(a_i\frac{(x_i - z)^{N+1}}{(N+1)!}\right) + \sum_{i=1}^{m} f^{(N+1)}(\eta_i)\left(b_i\frac{(y_i - z)^{N}}{N!}\right) \\ &+ \sum_{i=1}^{n}\left(\hat{f}(x_i) - f(x_i)\right)a_i + \sum_{i=1}^{m}\left(\hat{f}'(y_i) - f'(y_i)\right)b_i \quad . \end{aligned} \tag{6.3}$$

Let the "magnitude" $\beta > 0$ and the "roughness" $\gamma > 0$ be two parameters that describes the behavior of the function f. Determination of their values from the datapoints is discussed in Sections 6.2.4 and 6.2.5. For now, we consider these two parameters as given. We assume each $f^{(k)}(z), k = 1, \ldots, N$ to be a random variable with zero mean and standard deviation $\beta\gamma^k$. Assume that $f^{(N+1)}(\xi_i), i = 1, \ldots, n$ and $f^{(N+1)}(\eta_i), i = 1, \ldots, m$ are also random variables with standard deviation $\beta\gamma^{N+1}$. These random variables are assumed to be mutually independent, and independent of the measurement errors. Under

these assumptions, the expectation of the squared interpolation residual (6.3) is:

$$\begin{aligned} E\left[r(z)^2\right] = \beta^2 \sum_{k=1}^{N} \gamma^{2k} \left(\sum_{i=1}^{n} a_i \frac{(x_i - z)^k}{k!} + \sum_{i=1}^{m} b_i \frac{(y_i - z)^{k-1}}{(k-1)!} \right)^2 \\ + \beta^2 \sum_{i=1}^{n} \gamma^{2N+2} \left(a_i \frac{(x_i - z)^{N+1}}{(N-1)!} \right)^2 + \beta^2 \sum_{i=1}^{m} \gamma^{2N+2} \left(b_i \frac{(y_i - z)^N}{N!} \right)^2 \\ + \sum_{i=1}^{n} e_{a\,i}^2 a_i^2 + \sum_{i=1}^{m} e_{b\,i}^2 b_i^2 \quad , \end{aligned} \tag{6.4}$$

which is a quadratic form of the coefficients vector

$$[a\,b] = [a_1 \ldots a_n\, b_1 \ldots b_m].$$

In fact, let X^a be an $N \times n$ matrix with each matrix element

$$X^a_{ki} = \gamma^k \frac{(x_i - z)^k}{k!},$$

X^b be an $N \times m$ matrix with

$$X^b_{ki} = \gamma^k \frac{(y_i - z)^{k-1}}{(k-1)!},$$

G be an $n + m \times n + m$ diagonal matrix with

$$G_{ii} = \begin{cases} \gamma^{N+1} \dfrac{(x_i - z)^{N+1}}{(N+1)!}, & i = 1, \ldots, n \\ \gamma^{N+1} \dfrac{(y_{i-n} - z)^N}{N!}, & i = n+1, \ldots, n+m \end{cases},$$

and H also be an $n + m \times n + m$ diagonal matrix with

$$H_{ii} = \begin{cases} e_{a\,i}, & i = 1, \ldots, n \\ e_{b\,i-n}, & i = n+1, \ldots, n+m \end{cases},$$

the matrix form of the expectation of $r(z)^2$ can be written as

$$E\left[r(z)^2\right] = [a\,b] \left(\beta^2 \begin{bmatrix} X^{a\,\mathrm{T}} \\ X^{b\,\mathrm{T}} \end{bmatrix} \begin{bmatrix} X^a\, X^b \end{bmatrix} + \beta^2 G^2 + H^2 \right) [a\,b]^{\mathrm{T}}, \tag{6.5}$$

which is the quantity we want to minimize by choosing an optimal set of a and b, under the normalization constraint (6.2). Denote the symmetric semi-positive-definite matrix

$$A = \beta^2 \begin{bmatrix} X^{a\,\mathrm{T}} \\ X^{b\,\mathrm{T}} \end{bmatrix} \begin{bmatrix} X^a \, X^b \end{bmatrix} + \beta^2 G^2 + H^2,$$

and the length $n+m$ vector c as

$$c_i = \begin{cases} 1 & i = 1, \ldots, n \\ 0 & i = n+1, \ldots, n+m \; . \end{cases}$$

The minimization of the expectation of the squared interpolation residual (6.4) under the normalization constraint (6.2) becomes a quadratic programming problem

$$\min \, [a \, b] \, A \, [a \, b]^{\mathrm{T}}, \quad c \, [a \, b]^{\mathrm{T}} = 1. \tag{6.6}$$

Numerical methods for solving this quadratic programming for a and b is discussed in Section 6.2.6. Once a and b are calculated, the interpolant can be computed with (6.1), and the expected error of the interpolation can be computed using (6.5).

6.2.2 Continuity, smoothness and boundedness

In this section, we analyze the case when there is no uncertainty in the measurements, i.e.,

$$\hat{f}(x_i) = f(x_i), \qquad \hat{f}'(y_i) = f'(y_i).$$

Here, we first show that our interpolant $\tilde{f}$ is a bounded, continuous and infinitely differentiable rational function that goes through each datapoint:

$$\tilde{f}(x_i) = f(x_i), \quad i = 1, \ldots, n.$$

To show that $\tilde{f}$ is a rational function of z, we first analyze what values of z makes the quadratic programming problem (6.6) positive-definite, and what values of z makes it singular. We then derive a rational form of $\tilde{f}$ when the matrix A in (6.6) is positive-definite. Finally, we show that $\tilde{f}$ satisfies the same rational form when A is singular.

Lemma 6.2.1. *Let each $x_i \neq x_j$ for any $1 \leq i < j \leq n$, and each $y_i \neq y_j$ for any $1 \leq i < j \leq m$. Let X^a, X^b and G be the matrices described in the previous section, then*

$$A = \beta^2 \left(\begin{bmatrix} X^{a\,\mathrm{T}} \\ X^{b\,\mathrm{T}} \end{bmatrix} \begin{bmatrix} X^a & X^b \end{bmatrix} + G^2 \right)$$

is positive-definite when $z \neq x_i$ for any $i = 1, \ldots, n$; or rank $n + m - 1$ and positive-semi-definite when $z = x_i$ for some i.

Proof. To prove that A is positive-definite when $z \neq x_i$, we need to prove that $[a\,b]\,A\,[a\,b]^{\mathrm{T}} > 0$ whenever $a \neq 0$ or $b \neq 0$. We first consider the case when $a \neq 0$. Since $z \neq x_i$ for any i, the matrix first n diagonal elements of G are nonzero by definition. Therefore, $G[a\,b]^{\mathrm{T}} \neq 0$ when $a \neq 0$, and

$$[a\,b]\,A\,[a\,b]^{\mathrm{T}} \geq \beta^2 \left(G\,[a\,b]^{\mathrm{T}}\right)^{\mathrm{T}} \left(G\,[a\,b]^{\mathrm{T}}\right) > 0.$$

The other situation is $a = 0$ but $b \neq 0$. We further divide this into two sub-situations: when $z \neq y_i$ for any i, and when $z = y_i$ for some i. When $z \neq y_i$, all diagonal terms in G are nonzero, and $b \neq 0 \Longrightarrow G\,[a\,b]^{\mathrm{T}} \neq 0 \Longrightarrow [a\,b]\,A\,[a\,b]^{\mathrm{T}} > 0$. In the other case, $z = y_i$ for some i, the diagonal term in G corresponding to y_i is zero, and all other diagonal terms in G are nonzero. If $b_j \neq 0$ for some $j \neq i$, then $G\,[a\,b]^{\mathrm{T}} \neq 0$, and $[a\,b]\,A\,[a\,b]^{\mathrm{T}} > 0$ follows. Otherwise, b_i is the only nonzero element in b, and $G\,[a\,b]^{\mathrm{T}} = 0$. But since the first row of X^b is all 1, the first element of $X^b\,b^{\mathrm{T}}$ is equal to b_i in this case. Therefore, $X^b\,b^{\mathrm{T}} \neq 0$, and

$$[a\,b]\,A\,[a\,b]^{\mathrm{T}} \geq \beta^2 \left([X^a\,X^b]\,[a\,b]^{\mathrm{T}}\right)^{\mathrm{T}} \left([X^a\,X^b]\,[a\,b]^{\mathrm{T}}\right) = \beta^2 \left(X^b\,b^{\mathrm{T}}\right)^{\mathrm{T}} \left(X^b\,b^{\mathrm{T}}\right) > 0.$$

We have proven that A is positive-definite when $z \neq x_i$ for any i.

Now we prove the statement when $z = x_i$ for some i. In general,

$$[a\,b]\,A\,[a\,b]^{\mathrm{T}} = \beta^2 \left([X^a\,X^b]\,[a\,b]^{\mathrm{T}}\right)^{\mathrm{T}} \left([X^a\,X^b]\,[a\,b]^{\mathrm{T}}\right) + \beta^2 \left(G\,[a\,b]^{\mathrm{T}}\right)^{\mathrm{T}} \left(G\,[a\,b]^{\mathrm{T}}\right) \geq 0,$$

so A is always positive-semi-definite. To prove that it is rank $n+m-1$ when $z = x_i$, we only need to show that its nullspace is 1-D. Specifically, we show that the basis of its nullspace is $[a^{bi}\,b^{bi}]$, where

$$a_j^{bi} = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases} \qquad \text{and} \qquad b^{bi} = 0.$$

In fact, since $z = x_i$, the ith diagonal elements of G is 0, and its other diagonal element $G_{jj} \neq 0$ when $j \neq i, j = 1, \ldots, n$. Therefore,

$$G\,[a^{bi}\,b^{bi}]^{\mathrm{T}} = 0.$$

Also, the entire ith column of the matrix X^a is 0 when $z = x_i$, so

$$[X^a\,X^b]\,[a^{bi}\,b^{bi}]^{\mathrm{T}} = 0$$

as well, and it follows that

$$[a^{bi}\,b^{bi}]\,A\,[a^{bi}\,b^{bi}]^{\mathrm{T}} = 0.$$

This proves that $[a^{bi}\,b^{bi}]$ is in the null space of A. On the other hand, if $[a\,b]$ is not spanned by $[a^{bi}\,b^{bi}]$, then either $a_j \neq 0$ for some $j \neq i$ or $b \neq 0$. In the first case, since the jth diagonal element of G is nonzero, $G\,[a\,b] \neq 0$. Therefore,

$$[a\,b]\,A\,[a\,b]^{\mathrm{T}} \geq \beta^2 \left(G\,[a\,b]^{\mathrm{T}}\right)^{\mathrm{T}} \left(G\,[a\,b]^{\mathrm{T}}\right) > 0.$$

In the other case, $b \neq 0$. We consider the two cases: $z \neq y_i$ for all i, and $z = y_i$ for some i. By the same arguments, we have $[a\,b]\,A\,[a\,b]^{\mathrm{T}} > 0$ for both cases. In conclusion, $[a\,b]\,A\,[a\,b]^{\mathrm{T}} > 0$ if and only if $[a\,b]$ is not spanned by $[a^{bi}\,b^{bi}]$. Therefore, the $n+m \times n+m$ matrix A is rank $n+m-1$ and positive-semi-definite when $z = x_i$ for some i. □

In this section, we assume the measurement error to be zero, and the matrix $H = 0$. Therefore, this lemma implies that the matrix A is positive-definite whenever the interpolation point z does not overlap with any value node. In this case, the Lagrangian of the quadratic programming is

$$\min\,[a\,b]\,A\,[a\,b]^{\mathrm{T}} - 2\lambda\,c\,[a\,b]^{\mathrm{T}}.$$

By taking the gradient of the Lagrangian, the solution of the quadratic programming is

$$A\,[a\,b]^{\mathrm{T}} = \lambda\,c^{\mathrm{T}}, \quad c\,[a\,b]^{\mathrm{T}} = 1,$$

which can be obtained by solving the linear system

$$\begin{aligned} A\,[a^* b^*]^{\mathrm{T}} &= c^{\mathrm{T}}, \\ [a\,b] &= \frac{[a^* b^*]}{c\,[a^* b^*]^{\mathrm{T}}}\ . \end{aligned} \tag{6.7}$$

Let $A^* = |A|A^{-1}$ be the adjugate matrix of A, each element of the adjugate matrix A^*_{ij} is a cofactor of A. From (6.7), we have $[a^* b^*]^{\mathrm{T}} = A^{-1}c^{\mathrm{T}}$, where $A^{-1} = A^*/|A|$. Therefore,

$$a^*_i = \sum_{j=1}^{n} \frac{A^*_{i\,j}}{|A|}, \quad b^*_i = \sum_{j=1}^{n} \frac{A^*_{i+n\,j}}{|A|}.$$

Since $c\,[a^* b^*]^{\mathrm{T}} = \sum_{j=1}^{n} a^*_j$,

$$a_i = a^*_i \left/ \sum_{j=1}^{n} a^*_j = \sum_{j=1}^{n} A^*_{i\,j} \right/ \sum_{k=1}^{n}\sum_{j=1}^{n} A^*_{k\,j}\ , \quad b_i = b^*_i \left/ \sum_{j=1}^{n} a^*_j = \sum_{j=1}^{n} A^*_{i+n\,j} \right/ \sum_{k=1}^{n}\sum_{j=1}^{n} A^*_{k\,j}\ .$$

With the interpolation coefficients in this form, the interpolant (6.1) is

$$\tilde{f}(z) = \left(\sum_{i=1}^{n} f(x_i) \sum_{j=1}^{n} A^*_{i\,j} + \sum_{i=1}^{m} f'(y_i) \sum_{j=1}^{n} A^*_{i+n\,j} \right) \Bigg/ \left(\sum_{i=1}^{n}\sum_{j=1}^{n} A^*_{i\,j} \right)\ . \tag{6.8}$$

Since each element of the matrix A is a polynomial of the interpolation point z of degree $N+1$ or less, each of its cofactor A^*_{ij} is a polynomial of z of degree $(N+1)(n+m-1)$ or less. From (6.8), we conclude that when A is positive-definite, $\tilde{f}$ is a rational function of z whose denominator and numerator are both polynomials of order $(N+1)(n+m-1)$ or less. This rational form of our interpolant is obtained by assuming that z does not equal to any given value node x_i.

Lemma 6.2.1 indicates that the only case in which the matrix A in the quadratic programming problem (6.6) is not positive-definite is when $z = x_i$ for some $1 \le i \le n$. In this case, the ith column of X^a as well as the ith diagonal element of G is zero, and the matrix A is only positive-semi-definite with a 1-D nullspace. Therefore, the minimum of the quadratic form (6.6) is 0 when $a_i = 1, a_j = 0, j \neq i$ and $b_j = 0, j = 1, \ldots, m$, which is the unique solution of the quadratic programming problem. Our interpolant function with this a and b is

$$\tilde{f}(x_i) = f(x_i)\ .$$

On the other hand, due to the zero elements in the ith column in X and G, both the ith column and the ith row of the matrix A is zero, which makes A^*_{ii} the only nonzero cofactor of A. Incorporate this into (6.8), we get

$$\tilde{f}(z) = \frac{f(x_i)A^*_{ii}}{A^*_{ii}} = f(x_i).$$

This means that equation (6.8) is a universal formula for our interpolant function irrespective to whether z is on one of the value nodes or not. This proves the following lemma:

Proposition 6.2.2. *The interpolant $\tilde{f}$ given by (6.1) and (6.6) can be represented in the form of (6.8), which is a rational function of order $(N+1)(n+m-1)$ or less both in its numerator and denominator.*

In order to show that this rational function is continuous and smooth, we only need to prove that its denominator $\left(\sum_{i=1}^{n}\sum_{j=1}^{n} A^*_{i\,j}\right)$ never takes the value of 0. In fact, when $z = x_i$, all cofactors other than A^*_{ii} are zero, and the denominator simply equals to A^*_{ii}, which is nonzero because A is only one rank deficient (Lemma 6.2.1). On the other hand, when z does not equal to any x_i, A is positive-definite. Therefore, A^* is also positive-definite, and the denominator

$$\sum_{i=1}^{n}\sum_{j=1}^{n} A^*_{i\,j} = c\,A^*\,c^{\mathrm{T}} > 0$$

where

$$c_i = \begin{cases} 1 & i = 1, \ldots, n \\ 0 & i = n+1, \ldots, n+m\ . \end{cases}$$

Therefore, the rational form (6.8) of our interpolant $\tilde{f}$ has no poles on the real axis $(-\infty, \infty)$, and the lemma below follows.

Proposition 6.2.3. *The interpolant $\tilde{f}$ given by (6.1) and (6.6) is continuous and infinitely differentiable.*

In addition to being a smooth analytic function, our interpolant is bounded. In fact, $\tilde{f}(z)$ converges to a finite limit as z approaches infinity.

Proposition 6.2.4. *As the interpolation point z approaches infinity on either side,*

$$\lim_{z\to\pm\infty} \tilde{f}(z) = \frac{1}{n}\sum_{i=1}^{n} f(x_i)\ .$$

Proof. Define diagonal matrix D, such that

$$D_{ii} = \begin{cases} \dfrac{(\gamma z)^{2N+2}}{(N+1)!^2}\ , & 1 \le i \le n \\ \dfrac{(\gamma z)^{2N}}{N!^2}\ , & n < i \le n+m\ . \end{cases}$$

From the definition of matrix A, one can verify that

$$\lim_{z\to\pm\infty} \left(D^{-1}A\right)_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j\ . \end{cases}$$

On the other hand, we can write (6.7) as

$$\left(D^{-1}A\right)[a^*b^*]^{\mathrm{T}} = D^{-1}c^{\mathrm{T}}.$$

Multiplying both sides by $(\gamma z)^{2N+2}/(N+1)!^2$ yields

$$\left(D^{-1}A\right)[a^{**}b^{**}]^{\mathrm{T}} = c^{\mathrm{T}},$$

where

$$[a^{**}, b^{**}] = \frac{(\gamma z)^{2N+2}}{(N+1)!^2}\,[a^*b^*].$$

As $z \to \infty$, $D^{-1}A$ converges to the identity matrix. The solution of the linear system $a^{**} \to 1$, $b^{**} \to 0$, and

$$\lim_{z\to\infty} a_i = \lim_{z\to\infty} \frac{a_i^*}{\sum_{i=1}^{n} a_i^*} = \lim_{z\to\infty} \frac{a_i^{**}}{\sum_{i=1}^{n} a_i^{**}} = \frac{1}{n},$$

$$\lim_{z\to\infty} b_i = \lim_{z\to\infty} \frac{b_i^*}{\sum_{i=1}^{n} a_i^*} = \lim_{z\to\infty} \frac{b_i^{**}}{\sum_{i=1}^{n} a_i^{**}} = 0.$$

Therefore,

$$\lim_{z\to\pm\infty} \tilde{f}(z) = \lim_{z\to\pm\infty} \left(\sum a_i f(x_i) + \sum b_i f'(y_i)\right) = \frac{1}{n}\sum_{i=1}^{n} f(x_i)\ .$$

□

This lemma shows that the interpolant function converges to the average of the function value at all the datapoints as z goes to infinity. Since $\tilde{f}$ is a rational function with no pole on the open real axis, this lemma shows that it also has no pole at ∞. Another significant implication of this lemma is that our interpolant is bounded in $(-\infty, \infty)$.

In addition to continuity, smoothness and boundedness, we analyze how the interpolant function matches the given datapoints.

Proposition 6.2.5. *$\tilde{f}$ goes through each the value datapoints*

$$\tilde{f}(x_i) = f(x_i), \quad i = 1, \ldots, n.$$

Moreover, if x_i is also one of the gradient nodes, i.e., $x_i = y_{i'}$, then $\tilde{f}$ matches the gradient at this point, i.e.,

$$\tilde{f}'(x_i) = f'(x_i).$$

Proof. We have already shown in proving Lemma 6.2.2 that when $z = x_i$, the minimum of the quadratic programming (6.6) is achieved when $a_i = 1, a_j = 0, j \neq i$ and $b_j = 0, j = 1, \ldots, m$, and by definition (6.1), the interpolant goes through the datapoint $(x_i, f(x_i))$, i.e., $\tilde{f}(x_i) = f(x_i)$. Here we show that if a value node x_i is also a gradient node, i.e., $x_i = y_{i'}$ for some $1 \leq i' \leq m$, our interpolant not only goes through this value datapoint, but also matches the gradient $f'(y_{i'})$.

Consider an interpolation point z in the vicinity of x_i, then

$$\epsilon = x_i - z$$

is small. The ith column of X^a is $\gamma^k \epsilon^k / k!$. In big O notation, we can write

$$X^a_{ki} = \begin{cases} \gamma\epsilon & k = 1 \\ O(\epsilon^2) & k > 1 \end{cases}.$$

Similarly, the i'th column of X^b is $\gamma^k \epsilon^{k-1}/(k-1)!$. In big O notation,

$$X^b_{ki'} = \begin{cases} \gamma & k = 1 \\ O(\epsilon) & k > 1 \end{cases}$$

Note that these two rows of X^a and X^b are almost linearly dependent when ϵ is small. In fact, denote $a^{\epsilon,i}$ and $b^{\epsilon,i}$ as

$$a_j^{\epsilon,i} = \begin{cases} 1 & j = i, \\ 0 & j \neq i, \end{cases} \qquad b_j^{\epsilon,i} = \begin{cases} -\epsilon & j = i', \\ 0 & j \neq i, \end{cases}$$

then

$$[X^a\, X^b]\, [a^{\epsilon,i} b^{\epsilon,i}]^{\mathrm{T}} = O(\epsilon^2).$$

On the other hand,

$$G_{ii} = \gamma^{N+1} \frac{\epsilon^{N+1}}{(N+1)!} = O(\epsilon^2), \quad G_{n+i'\, n+i'} = \gamma^{N+1} \frac{\epsilon^N}{N!} = O(\epsilon),$$

so

$$G\, [a^{\epsilon,i} b^{\epsilon,i}]^{\mathrm{T}} = O(\epsilon^2).$$

Therefore, the quadratic form in (6.6) is

$$\begin{aligned} [a^{\epsilon,i} b^{\epsilon,i}]\, A\, [a^{\epsilon,i} b^{\epsilon,i}]^{\mathrm{T}} =& \beta^2 \left([X^a\, X^b]\, [a^{\epsilon,i} b^{\epsilon,i}]^{\mathrm{T}} \right)^{\mathrm{T}} \left([X^a\, X^b]\, [a^{\epsilon,i} b^{\epsilon,i}]^{\mathrm{T}} \right) \\ &+ \beta^2 \left(G\, [a^{\epsilon,i} b^{\epsilon,i}]^{\mathrm{T}} \right)^{\mathrm{T}} \left(G\, [a^{\epsilon,i} b^{\epsilon,i}]^{\mathrm{T}} \right) \\ =& O(\epsilon^4). \end{aligned}$$

Based on this fact, we now prove that $[a^{\epsilon,i} b^{\epsilon,i}]$ is "almost" the solution of the quadratic programming (6.6) when ϵ is small. To be rigorous, denote $[a^*\, b^*]$ as the solution of (6.6), then we will prove that

$$[a^*\, b^*] = [a^{\epsilon,i} b^{\epsilon,i}] + O(\epsilon^2).$$

Once we have this, we can prove this lemma by

$$\begin{aligned} \tilde{f}'(z) &= \lim_{\epsilon \to 0} \frac{f(x_i) - \tilde{f}(z)}{\epsilon} \\ &= \lim_{\epsilon \to 0} \frac{f(x_i) - \left(\sum_{j=1}^n a_j^* f(x_j) + \sum_{j=1}^m b_j^* f'(y_j) \right)}{\epsilon} \\ &= \lim_{\epsilon \to 0} \frac{f(x_i) - \left(f(x_i) - \epsilon f'(y_{i'}) + O(\epsilon^2) \right)}{\epsilon} \\ &= f'(y_{i'})\ . \end{aligned}$$

To prove that the difference between $[a^*\, b^*]$ and $[a^{\epsilon,i} b^{\epsilon,i}]$ is $O(\epsilon^2)$, we use the fact that $[a^*\, b^*]$ minimizes $[a\, b]\, A\, [a\, b]^{\mathrm{T}}$ under the constraint $c[a\, b]^{\mathrm{T}}$, which is satisfied by $[a^{\epsilon,i} b^{\epsilon,i}]$. Therefore,

$$[a\, b]\, A\, [a\, b]^{\mathrm{T}} \leq [a^{\epsilon,i} b^{\epsilon,i}]\, A\, [a^{\epsilon,i} b^{\epsilon,i}]^{\mathrm{T}} = O(\epsilon^4)\ .$$

On the other hand,

$$[a\, b]\, A\, [a\, b]^{\mathrm{T}} \geq \beta^2 \left(G\, [a^* b^*]^{\mathrm{T}}\right)^{\mathrm{T}} \left(G\, [a^* b^*]^{\mathrm{T}}\right),$$

which implies that

$$G\, [a^* b^*]^{\mathrm{T}} = O(\epsilon^2).$$

As $\epsilon \to 0$, all diagonal elements of G are finite except for the ith and $n+i'$th. This leads to

$$a_j^* = O(\epsilon^2), \quad j \neq i \quad \text{and} \quad b_j^* = O(\epsilon^2), \quad j \neq i'.$$

In addition, since $\sum_{j=1}^{n} a_j^* = 1$, we have

$$a_i^* = 1 + O(\epsilon^2).$$

With this result, we have

$$a^* = a^{\epsilon,i} + O(\epsilon^2).$$

We now consider the first element of $[X^a\, X^b][a^*\, b^*]^{\mathrm{T}}$

$$\left[[X^a\, X^b][a^*\, b^*]^{\mathrm{T}}\right]_1 = \sum_{j=1}^{n} X_{1j}^a a_j^* + \sum_{j=1}^{m} X_{1j}^b b_j^* = \gamma\epsilon a_i^* + \gamma b_{i'}^* + O(\epsilon^2)\ .$$

On the other hand,

$$O(\epsilon^4) \geq [a\, b]\, A\, [a\, b]^{\mathrm{T}} \geq \beta^2 \left([X^a\, X^b]\, [a^* b^*]^{\mathrm{T}}\right)^{\mathrm{T}} \left([X^a\, X^b]\, [a^* b^*]^{\mathrm{T}}\right),$$

and

$$O(\epsilon^2) = \left[[X^a\, X^b][a^*\, b^*]^{\mathrm{T}}\right]_1.$$

So

$$\gamma\epsilon a_i^* + \gamma b_{i'}^* = O(\epsilon^2),$$

and

$$b^*_{i'} = O(\epsilon^2) - \epsilon(1 + O(\epsilon^2)) = -\epsilon + O(\epsilon^2).$$

With this result, we also have

$$b^* = b^{\epsilon,i} + O(\epsilon^2),$$

which concludes the proof. □

By summarizing the results of Propositions 6.2.2, 6.2.3, 6.2.4 and 6.2.5, we conclude this section with the following result.

Corollary 6.2.6. *The interpolant function $\tilde{f}(z)$ given by (6.1) and (6.6) is a rational function with no poles in $[-\infty, \infty]$, therefore is bounded, continuous and infinitely differentiable. $\tilde{f}$ goes through each value datapoint, and matches the gradient at a value datapoint if it is also a gradient datapoint.*

6.2.3 Approximation Error

This section discusses the approximation error, also known as the residual of the interpolation,

$$r(z) = \tilde{f}(z) - f(z).$$

We assume that the measurement errors are 0, in which case $\tilde{f}$ is a true interpolant that goes through each datapoint. The approximation error (6.3) can be rewritten as

$$r(z) = \sum_{k=1}^{N} w_k f^{(k)}(z) + \sum_{i=1}^{n} w_{N+1}^{\xi_i} f^{(N+1)}(\xi_i) + \sum_{i=1}^{m} w_{N+1}^{\eta_i} f^{(N+1)}(\eta_i),$$

where

$$\begin{aligned}
w_k &= \sum_{i=1}^{n} a_i \frac{(x_i - z)^k}{k!} + \sum_{i=1}^{m} b_i \frac{(y_i - z)^{k-1}}{(k-1)!}\,, && k = 1, \ldots, N \\
w_{N+1}^{\xi_i} &= a_i \frac{(x_i - z)^{N+1}}{(N+1)!}\,, && i = 1, \ldots, n \\
w_{N+1}^{\eta_i} &= b_i \frac{(y_i - z)^{N}}{N!}\,, && i = 1, \ldots, m\,.
\end{aligned}$$

It is clear that the approximation error of $\tilde{f}$ is a weighted sum of the derivatives of f of various orders. This is a distinct feature and an important advantage of our interpolation scheme. Traditional high-order interpolation schemes, such as Lagrange polynomial

interpolation, have a residual that depends solely on the high-order derivatives of f (Boyd, 1999). But even for smooth functions, the derivatives can grow exponentially as the order increases. This causes the interpolation residual to grow as more datapoints are added, unless one uses a specifically designed grid (Runge, 1901). In contrast, our interpolation scheme models the exponential growth of the derivatives of f, and chooses the interpolation coefficients a and b to balance the contribution of various derivatives to the residual. The parameter γ models the rate of growth or decrease of the derivatives of f, and determines the contribution of each derivative. The coefficients a and b, controlling the weights w_k of the derivatives in the composition of the residual, are chosen to explicitly minimize the modeled residual (6.4). This strategy makes our interpolation scheme immune to divergence on an arbitrary grids, if γ^k is a valid model of the growth of derivatives of f.

The "roughness" parameter plays a key role in determining the weights of different derivatives in the composition of the residual. From (6.4), these weights of the derivatives are bounded by

$$|w_k| \leq \frac{\sqrt{E\left[r(z)^2\right]}}{\beta\gamma^k}, \qquad k = 1, \ldots, N$$

$$|w_{N+1}^{\xi_i}| \leq \frac{\sqrt{E\left[r(z)^2\right]}}{\beta\gamma^{N+1}}, \qquad i = 1, \ldots, n$$

$$|w_{N+1}^{\eta_i}| \leq \frac{\sqrt{E\left[r(z)^2\right]}}{\beta\gamma^{N+1}}, \qquad i = 1, \ldots, m$$

These bounds of the weights grow or decrease at a rate of γ^{-k}. The larger γ is, the larger are the weights of low-order derivatives compared to the weights of high-order derivatives. When γ is small, the interpolation error originates mostly from high-order derivatives. In this case, numerical experiments show that the interpolant behaves like Lagrange polynomial interpolation: It is very accurate for functions whose high-order derivatives diminish rapidly, but may produce very large errors for functions with high-frequency components. On the other hand, when γ is large, the interpolant behaves more like Shepards' method (Gordon & Wixom, 1978): it has poor polynomial order of accuracy, but is extremely stable even for non-smooth functions. This dependence on γ is discussed in more detail in Section 6.2.5. In practice, it is beneficial to choose γ so that it is neither too small nor too large, so that the interpolant is both highly accurate and stable. A method of choosing γ in order to achieve this balance is also discussed in Section 6.2.5.

6.2.4 Determination of β and its influence on the interpolant

In the mathematical formulation of the interpolation scheme, we assume two given parameters that describes the behavior of the target function f: they are the "magnitude" β and the "roughness" γ. These parameters determine the standard deviation of the derivatives of f as random variables, i.e.,

$$\mathrm{Var}\left[f^{(k)}(z)\right]^{\frac{1}{2}} = \beta\gamma^k, \quad k = 0, 1, \ldots \tag{6.9}$$

In this section, we focus on these two parameters. We discuss how to estimate β and γ from the given datapoints, and how each of these two parameters affect the behavior of the interpolant.

We first consider β. It is called the magnitude of the target function f because it models its standard deviation

$$\mathrm{Var}\left[f(z)\right]^{\frac{1}{2}} = \beta.$$

Based on this fact, we can estimate β by taking the sample standard deviation of the given datapoints, i.e.,

$$\beta \approx \sqrt{\frac{\sum_{i=1}^{n}(f(x_i) - \bar{f}\,)^2}{n-1}}\,, \quad \text{where } \bar{f} = \frac{\sum_{i=1}^{n} f(x_i)}{n}\,.$$

In the presence of measurement error, an unbiased estimator for β is

$$\beta \approx \sqrt{\frac{\sum_{i=1}^{n}(\hat{f}(x_i) - \bar{f}\,)^2}{n-1} - \frac{\sum_{i=1}^{n} e_{a\,i}^2}{n}}\,, \quad \text{where } \bar{f} = \frac{\sum_{i=1}^{n} \hat{f}(x_i)}{n}\,.$$

assuming proper independence between the values of f and the measurement errors. However, this formula is not practical because it can give imaginary values of β. We therefore use

$$\beta \approx \sqrt{\frac{\sum_{i=1}^{n}(\hat{f}(x_i) - \bar{f}\,)^2}{n-1}\ \exp\left(-\frac{n-1}{n}\frac{\sum_{i=1}^{n} e_{a\,i}^2}{\sum_{i=1}^{n}(\hat{f}(x_i) - \bar{f}\,)^2}\right)}\,. \tag{6.10}$$

Using this formula, β is always a positive real number. When the measurement error is relatively small, it agrees with the unbiased estimator to first-order accuracy. Therefore, (6.10) offers an engineering compromise between bias and robustness.

β augments the relative importance of the given measurements to the importance of the

measurement errors, and determines how hard the interpolant tries to fit the data points. We first observe from the definition of A that β has no effect on the interpolant when $H = 0$, i.e., when there are no measurement errors. When measurement errors are present, the ratio of $e_{a\,i}$ to β presents a relation, between the contribution to the variation of $\hat{f}(x_i)$ from the measurement errors, and the contribution to the variation of $\hat{f}(x_i)$ from the variation of the function f itself. This can be seen from the definition of matrix A in the quadratic programming (6.6), as β determines the relative importance of the first two terms in A relative to the third term H, which corresponds to the measurement errors in $\hat{f}$. When β is small compared to the measurement errors, $A \approx H^2$, and the interpolant $\tilde{f}$ is a constant function whose value is

$$\tilde{f}(z) \equiv \sum_{i=1}^{n} \frac{1}{e_{a\,i}^2} \hat{f}(x_i).$$

In this case, the variability of the function values $\hat{f}(x_i)$ are attributed to the measurement errors, and $\tilde{f}$ makes no effort in trying to go through each datapoint. But when β is large compared to the measurement errors, the third term in the definition of A can be ignored compared to the first two terms. As discussed in the previous section, the interpolant $\tilde{f}$ goes through each datapoint $\hat{f}(x_i)$ when H is ignored. In this case, the variability of the function values $\hat{f}(x_i)$ is attributed to the variation of the function f itself, and the interpolant tries to go through each datapoint exactly. Similarly, the ratio of $e_{b\,i}$ and β [1] presents a relation, between the contribution to the variation of $\hat{f}'(y_i)$ from the measurement errors, and the contribution to the variation of $\hat{f}'(y_i)$ from the variation of the function gradient f'. A larger β forces the interpolant to make a more vigorous attempt to match the function value and its gradient at the given datapoints.

6.2.5 Determination of γ and its influence on the interpolant

The other parameter, the "roughness" γ, models how fast the kth derivative of f grows as k increases. γ is called the "roughness" because if f is a sine wave of angular frequency γ, i.e., $f(x) = e^{i\gamma x}$, then γ is the rate of growth of its derivative, i.e., $f^{(k)}(x) = \gamma^k f(x)$. We use this as a model for general smooth functions by assuming that the magnitude of the kth derivative of f grows exponentially as γ^k. Specifically, the standard deviation of $f^{(k)}(z)$ is modeled as in (6.9). In this model, the parameter γ describes the frequency of the fastest varying mode of the function f, or the reciprocal of the smallest length scale of f. With

[1] To be more specific, the ratio of $e_{b\,i}$ to $\beta\gamma$.

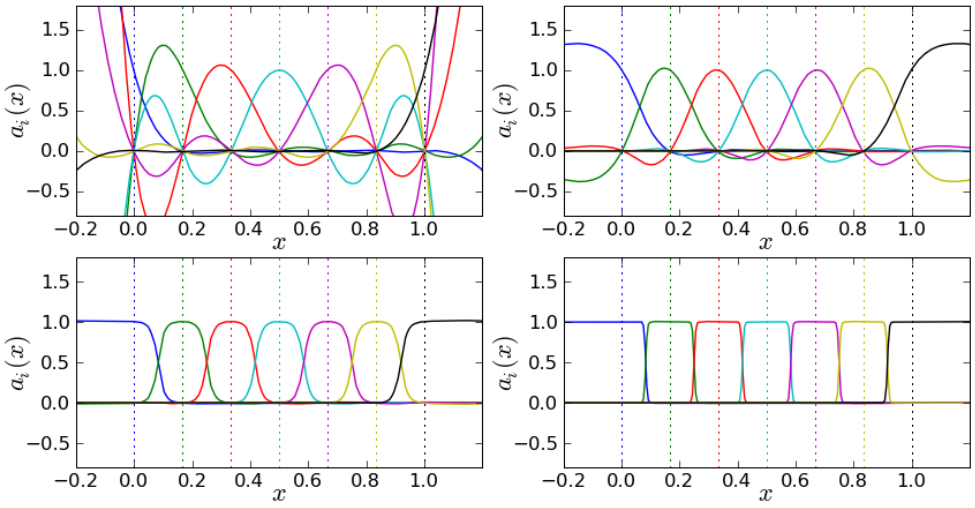

Figure 6.1: The basis functions on a uniform grid for $\gamma = 1$ (upper-left), $\gamma = 10$ (upper-right), $\gamma = 25$ (lower-left) and $\gamma = 100$ (lower-right). The dotted vertical lines indicate the location of the uniformly spaced value nodes, and each solid line of corresponding color is the unique interpolant that equals 1 at that node and 0 at all other nodes.

the appropriate γ, this model provides a valid estimate of the magnitude of $f^{(k)}(z)$ for most smooth functions. More importantly, with β obtained from (6.10), this model only has one free parameter γ to be determined from the often-limited information about f. This feature allows a robust estimate of the model parameter γ and contributes to the stability of our interpolation scheme.

We determine γ from the given datapoints using the bi-section method. Upper and lower bounds are first determined from the spacing of the nodes, then the interval of possible γ is bi-sected by interpolating each value datapoint using other datapoints, and comparing the actual residual $f(x_i) - \tilde{f}(x_i)$ with the expected residual calculated using (6.5). In determining the upper and lower bounds, we rely on the fact that the reciprocal of γ models the smallest length scale of f. On the other hand, the possible length scales that can be reconstructed from the finite number of datapoints are limited by the span of the datapoints on one end, and by the Nyquist sampling theorem on the other end. Specifically, we start the bi-section with

$$\gamma_{\min} = \frac{1}{\delta_{\max}}, \quad \gamma_{\max} = \frac{\pi}{\delta_{\min}},$$

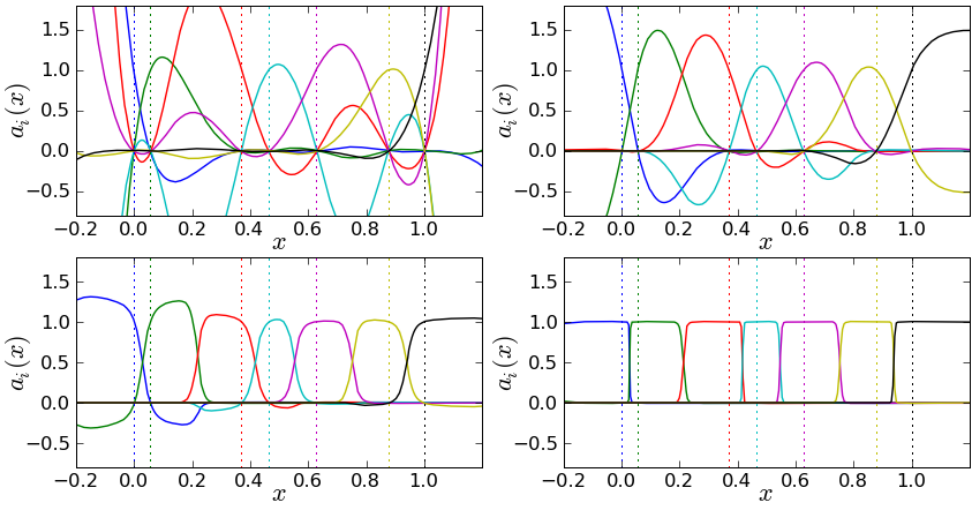

Figure 6.2: The basis functions on a non-uniform grid for different roughness γ: $\gamma = 1$ (upper-left), $\gamma = 10$ (upper-right), $\gamma = 25$ (lower-left) and $\gamma = 100$ (lower-right). The dotted vertical lines indicate the location of the non-uniformly spaced value nodes, and each solid line of corresponding color is the unique interpolant that equals 1 at that node and 0 at all other nodes.

where $\delta_{\max}$ is the maximum distance between any two nodes, including value nodes and gradient nodes. $\delta_{\min}$ is the minimum distance between any two nodes. The interval $[\gamma_{\min}, \gamma_{\max}]$ is then bi-sected logarithmically at each step by $\gamma_{\text{mid}} = \sqrt{\gamma_{\min}\gamma_{\max}}$. With this γ_{mid}, for each $i = 1, \ldots, n$, we use our interpolation scheme to calculate $\tilde{f}(x_i)$ with datapoints other than the one at x_i. We then compare the expected interpolation error estimated using (6.5) with the true residual $r(x_i) = \tilde{f}(x_i) - f(x_i)$. We decide that $\gamma < \gamma_{\text{mid}}$ if the expected residuals calculated by (6.5) are too large compared to the true residuals, or $\gamma > \gamma_{\text{mid}}$ if they are too small. This choice is based on the observation that a larger γ results in a more conservative error estimate. Specifically, we set

$$\gamma_{\max} = \gamma_{\text{mid}} \quad \text{if} \quad \frac{1}{n}\sum_{i=1}^{n} \frac{r(x_i)^2}{E\left[r(x_i)^2\right]} < 1 \,,$$

$$\gamma_{\min} = \gamma_{\text{mid}} \quad \text{if} \quad \frac{1}{n}\sum_{i=1}^{n} \frac{r(x_i)^2}{E\left[r(x_i)^2\right]} > 1 \,,$$

or when the measurement errors are nonzero,

$$\begin{aligned} \gamma_{\max} = \gamma_{\text{mid}} \quad \text{if} \quad \frac{1}{n}\sum_{i=1}^{n} \frac{\left(\tilde{f}(x_i) - \hat{f}(x_i)\right)^2}{E\left[r(x_i)^2\right] + e_{a\,i}^2} < 1 \ , \\ \gamma_{\min} = \gamma_{\text{mid}} \quad \text{if} \quad \frac{1}{n}\sum_{i=1}^{n} \frac{\left(\tilde{f}(x_i) - \hat{f}(x_i)\right)^2}{E\left[r(x_i)^2\right] + e_{a\,i}^2} > 1 \ . \end{aligned} \tag{6.11}$$

The bi-section continues until $\gamma_{\min}$ and $\gamma_{\max}$ are sufficiently close. We stop the bi-section when

$$\frac{\gamma_{\max}}{\gamma_{\min}} < T_\gamma$$

for some threshold T_γ. At this point, we use γ_{mid} as the estimation for the "roughness" parameter γ. Through numerical experiments with a number of different functions, we found that $T_\gamma \approx 1.1$ is enough to produce very good results.

The parameter γ determines how aggressive the interpolant tries to achieve polynomial accuracy. As discussed in Section 6.2.5, when γ is small, the approximation error mainly comes from high-order derivatives of f. In this case, the interpolation is accurate on smooth functions such as lower-order polynomials, but may produce very large errors if the function contains small length scales, which cause its high-order derivatives to grow rapidly. As γ gets large, the approximation error is dominated by low-order derivatives; the interpolation becomes more robust on oscillatory functions but less accurate on smooth functions. To illustrate the effects of γ, Figures 6.1 and 6.2 plot the basis of the interpolant for different values of γ on a uniform grid and a non-uniform grid. Both the uniform and non-uniform grid consist of 7 nodes. The basis of the interpolant at each node is defined as the unique interpolant on this grid that equals 1 on that node and 0 on all other nodes. An interpolant constructed from any function values given on this grid is a linear combination of these basis functions. Two important changes can be observed as γ varies. First, the support of the basis functions increases as γ decreases. Although our interpolation scheme is formally global, i.e., the function value at each data point influence the interpolant globally, the area where the influence is essentially nonzero is finite when γ is large. In both the uniform and non-uniform cases, the effective support of each basis function when $\gamma = 100$ barely covers the nearest neighborhood of the corresponding node. As γ decreases to 25, the support of each basis function extends to neighboring nodes, some times beyond a neighboring node

in the non-uniform grid. When γ further reduces to 10, the support of each basis covers multiple nodes. When $\gamma = 1$, the basis functions becomes global functions without finite support. The second change when γ decreases is the increase of the Lebesgue constant. The Lebesgue constant Λ is defined as the operator norm of the interpolation scheme as a linear mapping from the space of continuous functions to itself, i.e.,

$$\Lambda = \sup_{||f||=1} \left\| \tilde{f} \right\|,$$

where $\| \cdot \|$ is the maximum norm. It can be shown that Λ is equal to the maximum of all basis functions within the interpolation interval. Since the interpolant must go through each datapoint, the Lebesgue constant is greater or equal to 1. As can be seen from Figures 6.1 and 6.2, the basis functions are almost capped at 1 when $\gamma = 100$, and the Lebesgue constant is very close to unity. As γ decreases, the basis functions overshoot above 1, and the Lebesgue constant increases. When $\gamma = 1$, the Lebesgue constant is about 1.3 in the uniform grid, and above 2 in the non-uniform grid. We also notice that for the same γ, the Lebesgue constant is higher for the non-uniform grid. These two effects, the increase of the Lebesgue number, and the growth of the support of each basis function as γ decreases, dominate the behavior of the interpolant. A smaller γ generates a more global set of basis functions, allowing the use of a larger number of datapoints in the calculation of the interpolant value, which result in a more accurate approximation for smooth functions. A larger γ, on the other hand, represents a more conservative approach. By using fewer datapoints to determine the value of the interpolant, the interpolation scheme loses high-order accuracy for smooth functions; however, by constraining the basis functions to a more local support, it has a smaller Lebesgue constant, making it more robust for non-smooth functions.

To end this section, we further illustrate the role of γ by considering two extreme cases.

Proposition 6.2.7. *As the "roughness" parameter $\gamma \to 0^+$, the interpolant $\tilde{f}$ given by (6.1) and (6.6) converges pointwise to the Lagrange polynomial interpolant.*

Proposition 6.2.8. *As the "roughness" parameter $\gamma \to +\infty$, the interpolant $\tilde{f}$ given by (6.1) and (6.6) converges pointwise to the inverse distance weighting interpolant, a.k.a. Shepards' method, with power parameter $p = N + 1$.*

6.2.6 Numerical solution of the interpolation coefficients

We have derived the formulation of our interpolation scheme, the smoothness properties of the interpolant, a preliminary error analysis, and formulas to calculate the parameters β and γ from given datapoints. This section focuses on the numerical method for calculating the interpolant. Note that although we have proven that our interpolant function $\tilde{f}$ is a rational function (Proposition 6.2.2), we do not explicitly construct this rational function. Instead, the interpolation coefficients a and b are numerically calculated for each interpolation point z by solving the quadratic programming (6.6), and the interpolant is constructed point-by-point. We choose this approach because the rational form of $\tilde{f}$ is of very high-order. Both its numerator and denominator can be of order up to $(N+1)(n+m-1)$, where N is normally equal to $n+m$. Although it is theoretically possible to construct $\tilde{f}$ globally, it is prohibitively difficult to design a robust numerical algorithm to calculate its coefficients. For this reason, we resort to solving the quadratic programming (6.6) for a and b and calculate the value of the interpolant $\tilde{f}(z)$ at each z. Here we discuss the numerical issues that arise in solving this quadratic programming problem and our solution of these issues.

The main numerical difficulties result from the fact that the matrix A in the quadratic programming (6.6) is extremely ill-conditioned when N is large, rendering Cholesky decomposition, the most straightforward way of solving (6.7), highly unreliable. Two factors may contribute to the gigantic condition number of A. First, when $\gamma(x_i - z)$ and $\gamma(y_i - z)$ are small for multiple datapoints, the high-order terms in the corresponding columns of X^a and X^b are very small, making them almost linearly dependent, and the matrix $[X^a\, X^b]$ almost singular. At the same time, the corresponding diagonal terms in G are even smaller, making the regularization effect of G negligible. Therefore, when the corresponding measurement errors in H are also zero, A is almost singular. The second contribution to the condition number of A results from an opposite situation. When $\gamma(x_i - z)$ or $\gamma(y_i - z)$ is large, the high-order terms in the corresponding columns of X^a, X^b and corresponding diagonal elements in G are very large. As a result, the corresponding elements in A are much larger than the rest of the matrix, boosting its condition number. In practice, we find that these elements in A are often so large that they overflow the standard double-precision floating point numbers. These two factors combine to make the condition number of A so large that direct solution of the linear system is basically impossible. In fact, because A is very close to singular, Cholesky factorization fails most of the time, even though A is theoretically positive-definite. This requires the use of different strategies to treat these two respective

factors.

The first strategy is to avoid the explicit construction of A. Define

$$X = \beta[X^a \, X^b]$$

and diagonal matrix F such that

$$F_{ii}^2 = \beta^2 G_{ii}^2 + H_{ii}^2.$$

By definition of A, we have

$$A = X^{\mathrm{T}} X + F^{\mathrm{T}} F = \begin{bmatrix} X \\ F \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} X \\ F \end{bmatrix}.$$

By performing a QR decomposition

$$\begin{bmatrix} X \\ F \end{bmatrix} = QR, \tag{6.12}$$

we obtain R as the Cholesky factor of A:

$$A = R^{\mathrm{T}} Q^{\mathrm{T}} Q R = R^{\mathrm{T}} R.$$

The interpolation coefficients a and b then can be calculated by solving an upper triangular system and a lower triangular system. This procedure avoids the failure-prone Cholesky factorization, and improves numerical stability.

The second strategy is designed to solve the floating point overflow problem. We observe that due to the additional contribution from G and H, the diagonal elements of A are most prone to overflow. The elements in the solution vector $[a, b]$ corresponding to the overflowed diagonal elements of A are very close to zero. This is because γ is large enough so that the support of the corresponding basis function is finite, and z is far enough from the corresponding datapoint that it is outside the support of its basis function. In this case, there is no contribution to the interpolant at z from that datapoint, and the corresponding interpolation coefficient is zero. Based on this observation, our second strategy is to remove the rows and columns of A in which the diagonal elements overflow, solve the remaining

linear system for part of $[a\, b]$, and set the rest of $[a, b]$, whose corresponding matrix rows and columns are removed to 0.

These two strategies ensure the stability of our interpolation scheme for hundreds of datapoints and a wide range of γ. However, with very large number of datapoints and extreme values of γ, the system may be so ill-conditioned that our numerical scheme may still fail to produce an accurate result. If this does happen, an ultimate solution is to reduce the total order of the Taylor expansions N, so that $N < n + m$. Although this compromises the high-order accuracy of the interpolation scheme, it is often better than a solution corrupted by numerical error.

6.2.7 Numerical Examples

In this section, we apply our interpolation scheme to the following example functions:

1. A cosine wave $f(x) = \cos x$. This function is smooth and expected to be an easy case for interpolation schemes.
2. The Runge function $f(x) = \dfrac{1}{1 + x^2}$. It was used by Runge (Runge, 1901) to demonstrate the divergence of Lagrange interpolation on an equally spaced grid.
3. A cosine wave with a sharp Gaussian notch $f(x) = \cos x - 2e^{-(4x)^2}$. Since this function contains two distinct length scales, we use it as a simple model for multi-scale functions.
4. A discontinuous function $f(x) = \begin{cases} e^{-x^2}, & x > 0, \\ -e^{-x^2}, & x < 0. \end{cases}$

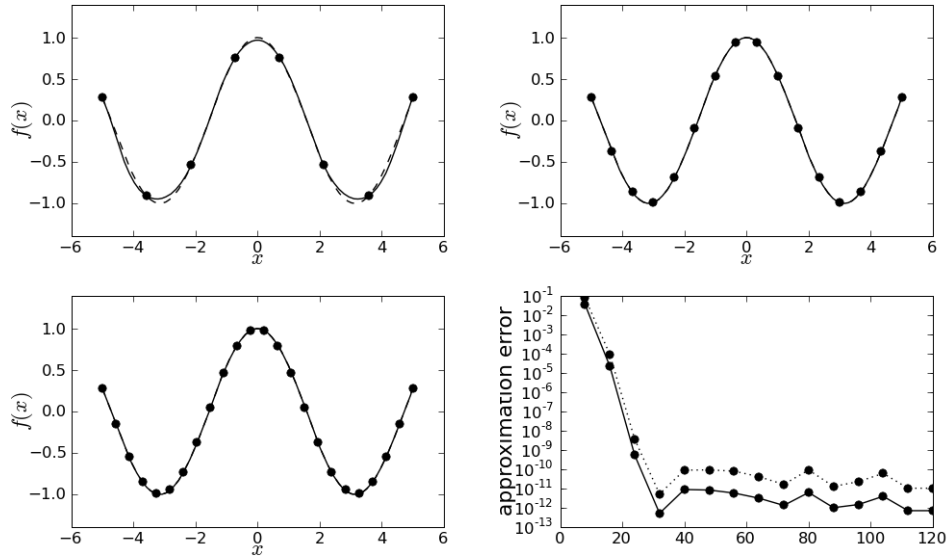

Figure 6.3: Interpolating the cosine wave using 8 (upper-left), 16 (upper-right), 24 (lower-left) uniform grid points. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

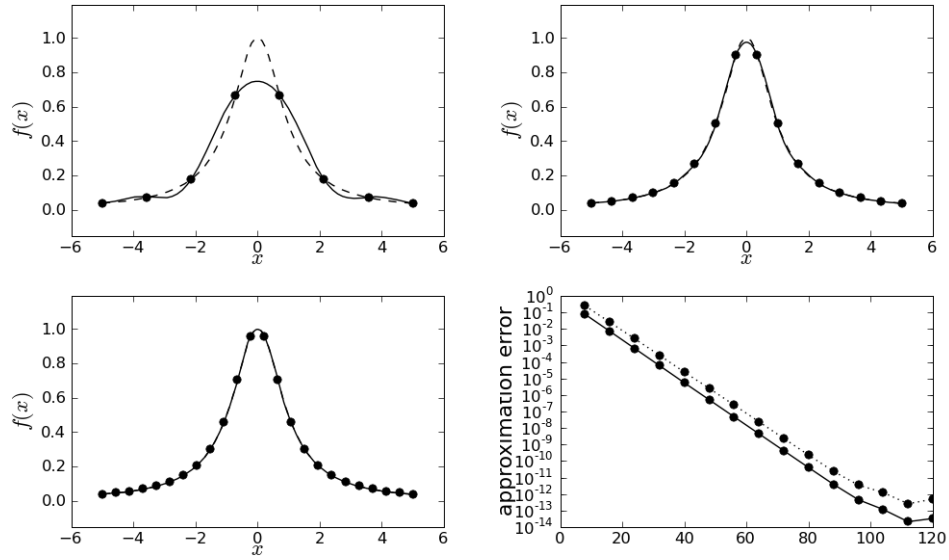

Figure 6.4: Interpolating the Runge function using 8 (upper-left), 16 (upper-right), 24 (lower-left) uniform grid points. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

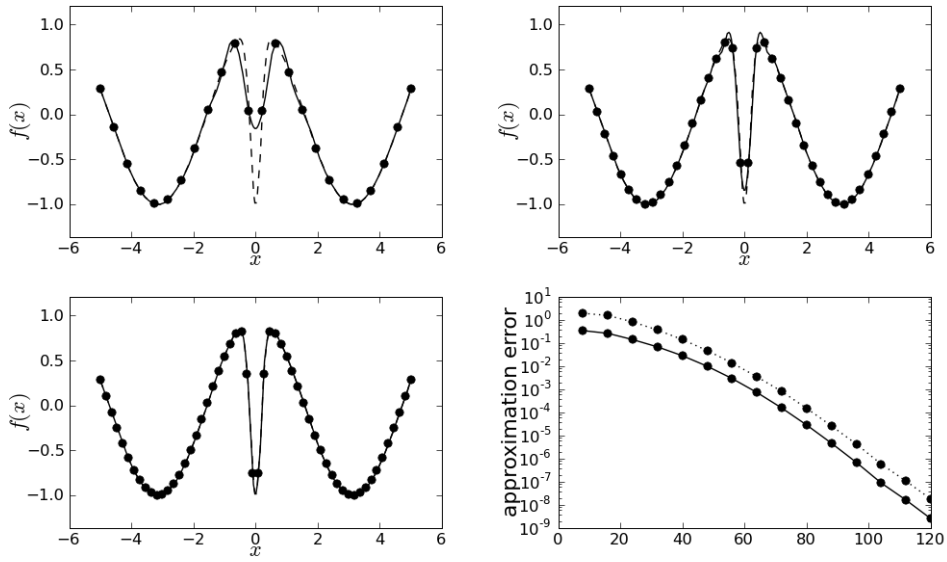

Figure 6.5: Interpolating the notched cosine function using 24 (upper-left), 40 (upper-right), 56 (lower-left) uniform grid points. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

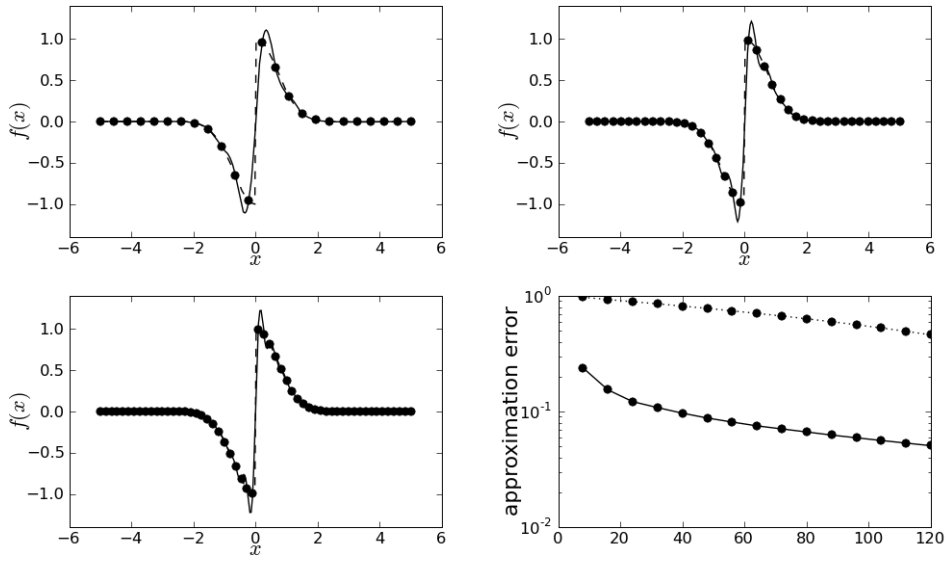

Figure 6.6: Interpolating the discontinuous function using 24 (upper-left), 40 (upper-right), 56 (lower-left) uniform grid points. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

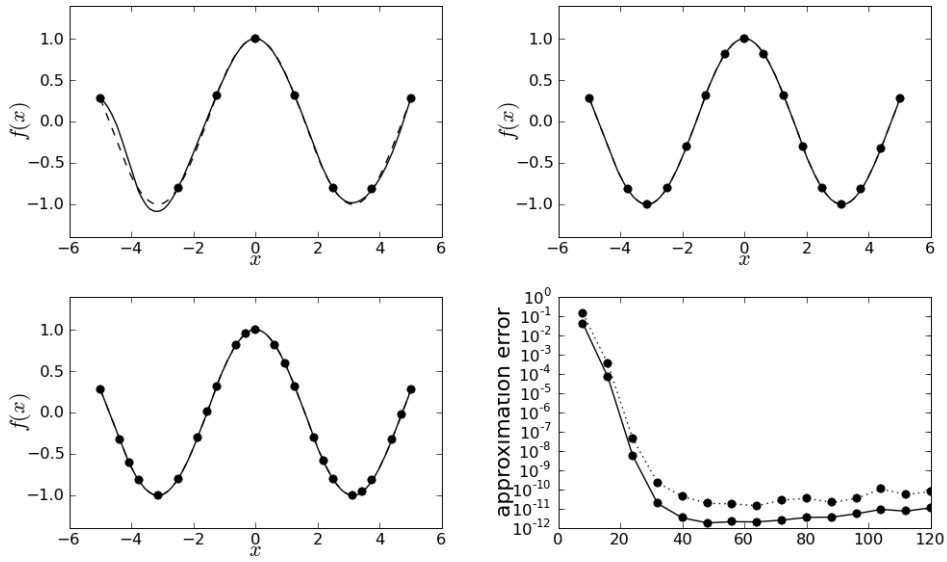

Figure 6.7: Interpolating the cosine wave using 8 (upper-left), 16 (upper-right), 24 (lower-left) quasi-random grid points. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

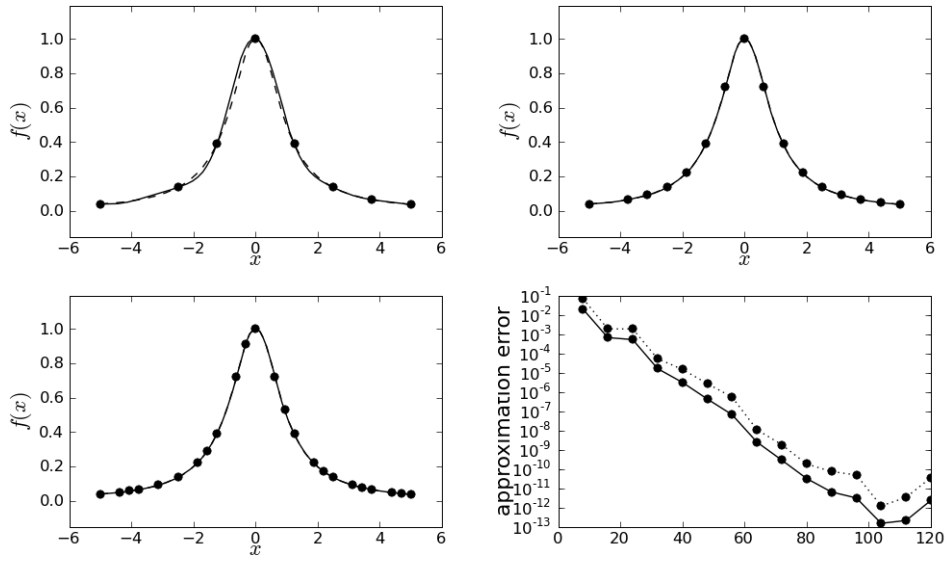

Figure 6.8: Interpolating the Runge function using 8 (upper-left), 16 (upper-right), 24 (lower-left) quasi-random grid points. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

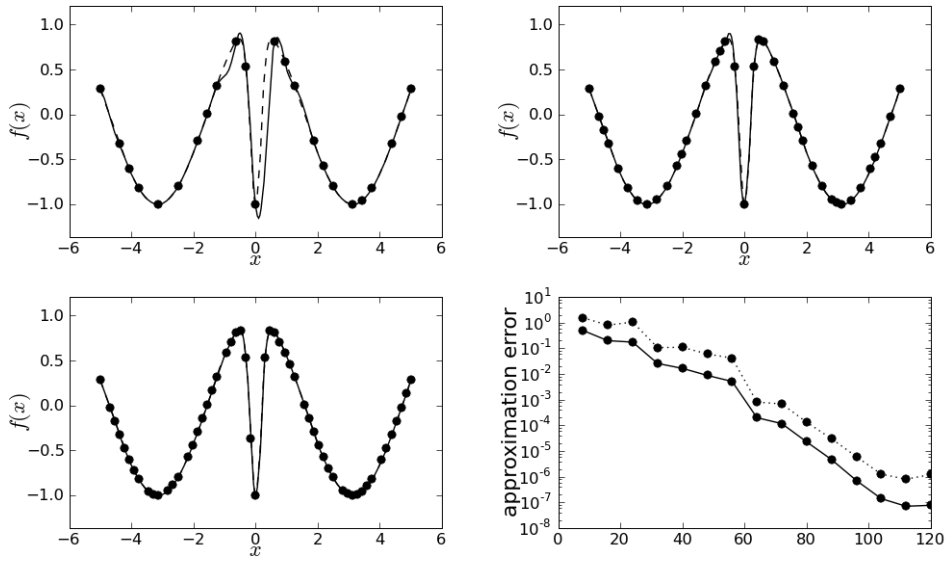

Figure 6.9: Interpolating the notched cosine function using 24 (upper-left), 40 (upper-right), 56 (lower-left) quasi-random grid points. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

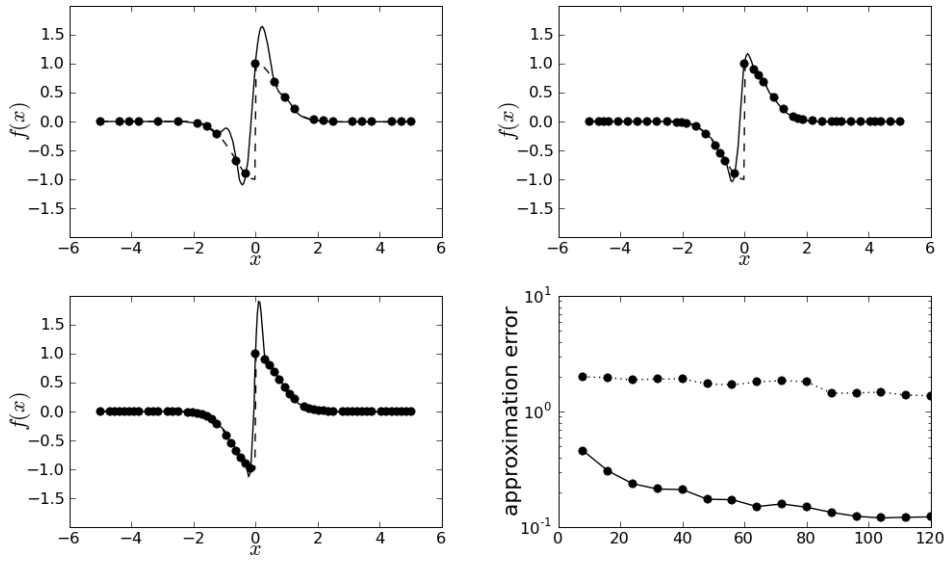

Figure 6.10: Interpolating the discontinuous function using 24 (upper-left), 40 (upper-right), 56 (lower-left) quasi-random grid points. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

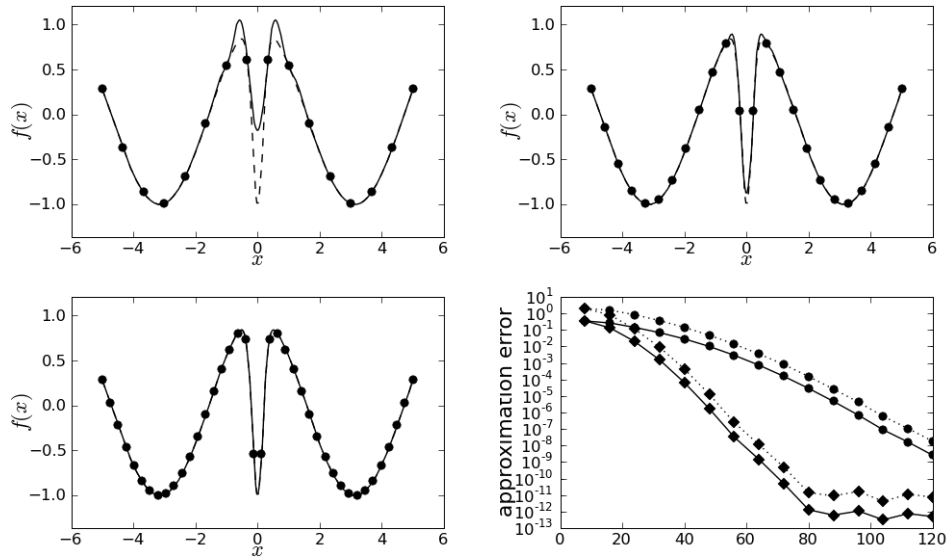

Figure 6.11: Interpolating the notched cosine function using 16 (upper-left), 24 (upper-right), 40 (lower-left) uniform grid with gradient. In the convergence plot (lower-right), the horizontal axis is the number of grid points; circles are error without gradient, diamonds are error with gradient.

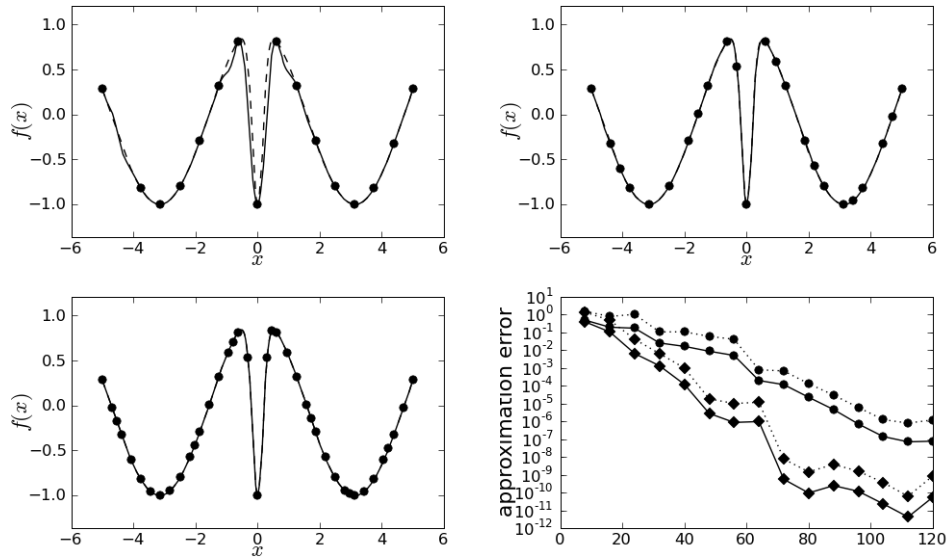

Figure 6.12: Interpolating the notched cosine function using 16 (upper-left), 24 (upper-right), 40 (lower-left) quasi-random grid with gradient. In the convergence plot (lower-right), the horizontal axis is the number of grid points; circles are error without gradient, diamonds are error with gradient.

For all these functions, the interpolant is constructed in the interval $[-5, 5]$ using two kinds of grids: a uniformly distributed grids, and a Niederreiter quasi-random sequence (Niederreiter, 1992). Figures 6.3 to 6.10 demonstrates the performance of our interpolation scheme on these four functions using both uniform and quasi-random grids. In these figures, all the datapoints are value datapoints, and no derivative information is used. In each of these figures, the first three plots show the target function as dashed lines and the interpolant approximation as solid lines. The dots indicate the location of the value datapoints. The fourth plot in each figure shows the convergence of our interpolations scheme. In these convergence plots, the horizontal axis is the number of datapoints used, and the vertical axis is the difference between the target function and the interpolant function, measured in L^∞ distance (dotted lines) and in L^2 distance (solid lines). As can be seen from these figures, our interpolation scheme works robustly for all four functions on both uniform and quasi-random grids. For the three smooth functions, it converges exponentially to a cutoff precision of approximately 10^{-10} to 10^{-13}. The rate of convergence is fastest for the cosine wave, and slowest for the multi-scale notched cosine function. This behavior is expected because the notched cosine function contains the most high-frequency components, while the plain cosine function contains the least. The cutoff precision is due to the round-off error accumulated in the QR factorization (6.12), and the solution of the linear systems with R. For the discontinuous function, we observe artificial overshoot and undershoot near the discontinuity, whose size doesn't seem to decrease as the grid refines. As a result, the L^∞ error in the convergence plots remains almost constant, and the L^2 error decreases slowly. Despite of this Gibbs-like phenomenon, the interpolant seems to converge pointwise to the target function, just as Lagrange interpolation does on a Lobatto grid. In these numerical experiments, our interpolation demonstrates high accuracy for smooth functions, and excellent robustness even for discontinuous functions.

While Figures 6.3 to 6.10 demonstrate our scheme with $m = 0$, Figures 6.11 to 6.12 shows the case with $m = n$ and $x_i = y_i$, i.e., each value datapoint is also a gradient datapoint. Uniform grids are used in Figure 6.11, and Niederreiter quasi-random grids are used in Figure 6.11. In the convergence plots, the horizontal axis is the number of value datapoints n, and the vertical axis is the error of the interpolant approximation. The circular dots show the L^2 (with solid lines) and L^∞ (with dotted lines) error of the interpolant constructed without gradient information, i.e., $m = 0$; the diamond symbols show the L^2 (with solid lines) and L^∞ (with dotted lines) error of the interpolant constructed with

gradient information at each value datapoint, i.e., $m = n$ and $y_i = x_i, i = 1, \ldots, n$. These two plots show that using derivative information on each datapoint greatly increases the accuracy of the interpolant approximation for the same number of value datapoints n. In both uniform grids and quasi-random grids, using derivative information essentially doubles the rate of convergence.

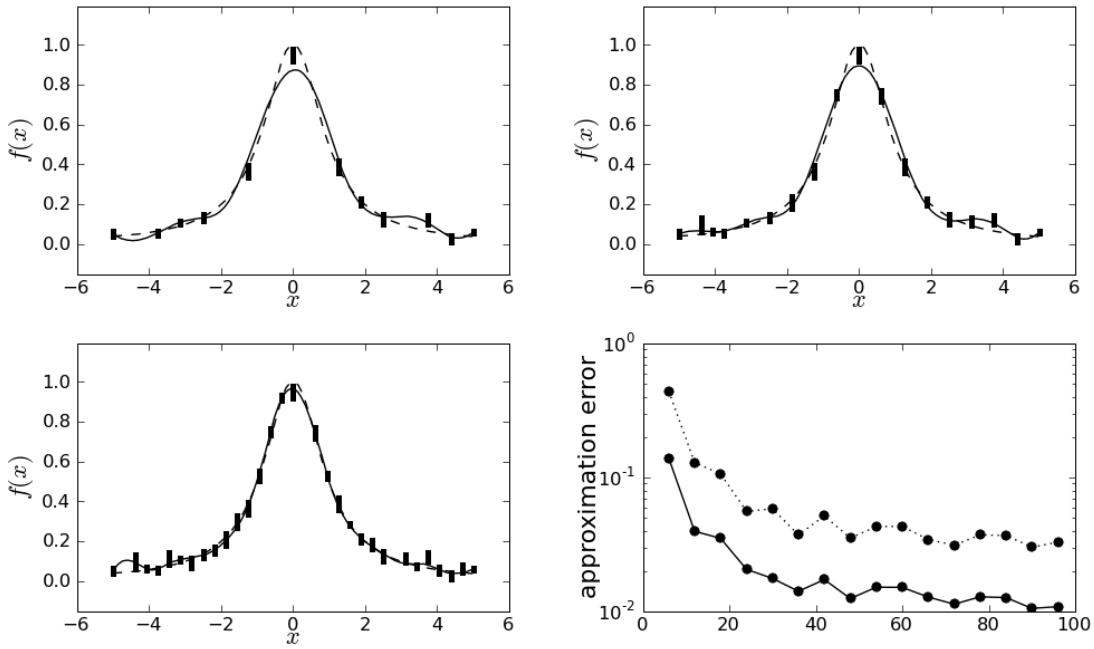

Figure 6.13: Interpolating the Runge function using 12 (upper-left), 18 (upper-right), 30 (lower-left) quasi-random grid with simulated measurement errors, whose magnitudes are indicated by the bars. In the convergence plot (lower-right), the horizontal axis is the number of grid points.

Figure 6.13 demonstrates our interpolation scheme when measurement errors are nonzero. Independent Gaussian random number are added to the function values to simulate measurement errors in the datapoints. In this case, our scheme becomes a nonlinear regression scheme by constructing a line that represents a "best fit" for the data. The convergence plot shows slow convergence to the target function, due to the corruption by the measurement errors.

6.3 Interpolation in multi-dimensional space

6.3.1 An extension of the multi-variate Taylor's theorem

Before we introduce our interpolation scheme in multi-dimensional space, we prove a generalized version of the multi-variate Taylor's theorem (to allow different orders of expansion in different directions). This extension allows us to expand Taylor series to different order in each dimension, greatly increasing the flexibility of our interpolation scheme in multi-dimensional space. We begin with two definitions.

Definition 6.3.1. Denote $\mathbb{N} = \{0, 1, \ldots\}$ as the set of non-negative integers. We call $S \subset \mathbb{N}^d$ an **expansion order set** if and only if S is finite, and for any $\kappa = (\kappa_1, \ldots, \kappa_d) \in S$ and $1 \leq k \leq d$ such that $\kappa_k > 0$,

$$\kappa - \mathbf{e}_k = (\kappa_1, \ldots, \kappa_{k-1}, \kappa_k - 1, \kappa_{k+1}, \ldots \kappa_d) \in S.$$

From the definition, we can see that any non-empty expansion order set contains the zero element $(0, \ldots, 0)$.

Definition 6.3.2. Let $S \subset \mathbb{N}^d$ be an expansion order set, the **boundary** of S, denoted as ∂S, is defined as

$$\partial S = \{\kappa \mid \kappa \notin S \text{ and } \kappa - \mathbf{e}_k \in S \text{ for some } 1 \leq k \leq d\} .$$

With these two definitions, the following theorem is an extension to the standard multi-variate Taylor's theorem, and will be used in the mathematical formulation of our multi-variate interpolation scheme.

Theorem 6.3.3. *Let $S \subset \mathbb{N}^d$ be a non-empty expansion order set, $\mathbf{a} = (a_1, \ldots, a_d)$ and $\mathbf{b} = (b_1, \ldots, b_d)$, then a d-dimensional smooth function f can be expanded as*

$$f(\mathbf{b}) = \sum_{\kappa \in S} \frac{f^{(\kappa)}(\mathbf{a})}{\kappa!} (\mathbf{b} - \mathbf{a})^{\kappa} + \sum_{\kappa \in \partial S} \zeta_S^{\kappa} \frac{f^{(\kappa)}(\mathbf{x}_\kappa)}{\kappa!} (\mathbf{b} - \mathbf{a})^{\kappa} \tag{6.13}$$

where each $\mathbf{x}_\kappa = \mathbf{b} + t_\kappa(\mathbf{a} - \mathbf{b})$, $0 < t_\kappa < 1$; $\kappa! = \prod_{k=1}^{d} \kappa_k!$ is the multi-variate factorial, and

$$\zeta_S^{\kappa} = \sum_{\substack{k=1 \\ \kappa - \mathbf{e}_k \in S}}^{d} \frac{\kappa_k}{|\kappa|} .$$

Proof. Denote

$$\mathbf{x}(t) = \mathbf{b} + t\,(\mathbf{a} - \mathbf{b}),$$

then $f(\mathbf{a}) = f(\mathbf{x}(1))$, $f(\mathbf{b}) = f(\mathbf{x}(0))$. We will prove the integral form of this theorem

$$f(\mathbf{b}) = \sum_{\kappa \in S} \frac{f^{(\kappa)}(\mathbf{a})}{\kappa!}(\mathbf{b} - \mathbf{a})^{\kappa} + \sum_{\kappa \in \partial S} \zeta_S^{\kappa} \int_0^1 \frac{f^{(\kappa)}(\mathbf{x}(t))}{\kappa!}(\mathbf{b} - \mathbf{a})^{\kappa}\, d\, t^{|\kappa|}, \tag{6.14}$$

where $|\kappa| = \sum_{k=1}^{d} \kappa_k$. This leads to (6.13) by the mean value theorem.

We use induction. The base case is when S contains only one member $(0, \ldots, 0)$. In this case, $\partial S = \{\mathbf{e}_k, k = 1, \ldots, d\}$, and $\zeta_S^{\mathbf{e}_k} = 1$ for each k. By the fundamental theorem of calculus and the chain rule, we have

$$f(\mathbf{b}) = f(\mathbf{a}) + \int_1^0 \frac{d\, f(\mathbf{x}(t))}{dt}\, dt = f(\mathbf{a}) + \sum_{\kappa \in \partial S} \int_0^1 f^{(\kappa)}(\mathbf{x}(t))(\mathbf{b} - \mathbf{a})^{\kappa}\, dt,$$

which is (6.14) for this base case.

Now for any expansion order set S, the hypothesis is that (6.14) holds true for any subset of S. Because S is finite, there exists a $\hat{\kappa} \in S$ such that $|\hat{\kappa}| \geq |\kappa|$ for any $\kappa \in S$. Note that $\hat{\kappa} + e_k \in \partial\Omega$ for any $1 \leq k \leq d$. Let $S' = \{\kappa \in S \mid \kappa \neq \hat{\kappa}\}$, then by the definition of expansion order set, $\hat{\kappa} - \mathbf{e}_k \in S'$ for any k such that $\hat{\kappa}_k > 0$. Therefore, we have $\hat{\kappa} \in \partial S'$ and

$$\zeta_{S'}^{\hat{\kappa}} = \sum_{\substack{k=1 \\ \hat{\kappa} - \mathbf{e}_k \in S'}}^{d} \frac{\hat{\kappa}_k}{|\hat{\kappa}|} = \sum_{\substack{k=1 \\ \hat{\kappa}_k > 0}}^{d} \frac{\hat{\kappa}_k}{|\hat{\kappa}|} = 1\ .$$

Integration by parts results in

$$\begin{aligned}
&\int_0^1 \frac{f^{(\hat{\kappa})}(\mathbf{x}(t))}{\hat{\kappa}!}(\mathbf{b}-\mathbf{a})^{\hat{\kappa}}\, d\, t^{|\hat{\kappa}|} \\
=&\frac{f^{(\hat{\kappa})}(\mathbf{a})}{\hat{\kappa}!}(\mathbf{b}-\mathbf{a})^{\hat{\kappa}} - \int_0^1 (\mathbf{b}-\mathbf{a})^{\hat{\kappa}}\, t^{|\hat{\kappa}|}\, d\frac{f^{(\hat{\kappa})}(\mathbf{x}(t))}{\hat{\kappa}!} \\
=&\frac{f^{(\hat{\kappa})}(\mathbf{a})}{\hat{\kappa}!}(\mathbf{b}-\mathbf{a})^{\hat{\kappa}} + \sum_{k=1}^{d}\int_0^1 \frac{f^{(\hat{\kappa}+\mathbf{e}_k)}(\mathbf{x}(t))}{\hat{\kappa}!}(\mathbf{b}-\mathbf{a})^{\hat{\kappa}+\mathbf{e}_k}\, t^{|\hat{\kappa}|}\, dt \\
=&\frac{f^{(\hat{\kappa})}(\mathbf{a})}{\hat{\kappa}!}(\mathbf{b}-\mathbf{a})^{\hat{\kappa}} + \sum_{k=1}^{d}\frac{\hat{\kappa}_k+1}{|\hat{\kappa}+\mathbf{e}_k|}\int_0^1 \frac{f^{(\hat{\kappa}+\mathbf{e}_k)}(\mathbf{x}(t))}{(\hat{\kappa}+\mathbf{e}_k)!}(\mathbf{b}-\mathbf{a})^{\hat{\kappa}+\mathbf{e}_k} dt^{|\hat{\kappa}+\mathbf{e}_k|} \\
=&\frac{f^{(\hat{\kappa})}(\mathbf{a})}{\hat{\kappa}!}(\mathbf{b}-\mathbf{a})^{\hat{\kappa}} + \sum_{\kappa\in\{\hat{\kappa}+\mathbf{e}_k, k=1,\ldots,d\}}\frac{\kappa_k}{|\kappa|}\int_0^1 \frac{f^{(\kappa)}(\mathbf{x}(t))}{\kappa!}(\mathbf{b}-\mathbf{a})^{\kappa} dt^{|\kappa|}\ .
\end{aligned}$$

Note that S' is also an expansion order set, and we can incorporate this integration by parts into the induction hypothesis for S'. Combining this with the fact that $S = S' \cup \{\hat{\kappa}\}$, we get

$$\begin{aligned}
f(\mathbf{b}) =& \sum_{\kappa\in S'}\frac{f^{(\kappa)}(\mathbf{a})}{\kappa!}(\mathbf{b}-\mathbf{a})^{\kappa} + \sum_{\kappa\in\partial S'}\zeta_{S'}^{\kappa}\int_0^1 \frac{f^{(\kappa)}(\mathbf{x}(t))}{\kappa!}(\mathbf{b}-\mathbf{a})^{\kappa}\, d\, t^{|\kappa|} \\
=& \sum_{\kappa\in S}\frac{f^{(\kappa)}(\mathbf{a})}{\kappa!}(\mathbf{b}-\mathbf{a})^{\kappa} + \sum_{\kappa\in\partial S'\setminus\{\hat{\kappa}\}}\zeta_{S'}^{\kappa}\int_0^1 \frac{f^{(\kappa)}(\mathbf{x}(t))}{\kappa!}(\mathbf{b}-\mathbf{a})^{\kappa}\, d\, t^{|\kappa|} \\
&+ \sum_{\kappa\in\{\hat{\kappa}+\mathbf{e}_k, k=1,\ldots,d\}}\frac{\kappa_k}{|\kappa|}\int_0^1 \frac{f^{(\kappa)}(\mathbf{x}(t))}{\kappa!}(\mathbf{b}-\mathbf{a})^{\kappa} dt^{|\kappa|}\ .
\end{aligned}$$

We complete the proof by unifying the previous two summations into a single summation over ∂S. From the definition of $\hat{\kappa}$ and S', we see that

$$\partial S = \left(\partial S' \setminus \{\hat{\kappa}\}\right) \cup \{\hat{\kappa}+\mathbf{e}_k, k=1,\ldots,d\}\ .$$

In addition, if $\kappa \notin \{\hat{\kappa}+\mathbf{e}_k, k=1,\ldots,d\}$, then $\kappa-\mathbf{e}_k \in S \iff \kappa-\mathbf{e}_k \in S'$. Otherwise, if $\kappa = \hat{\kappa}+\mathbf{e}_k$, then $\kappa-\mathbf{e}_{k'} \in S \iff \kappa-\mathbf{e}_{k'} \in S'$ or $k'=k$. We have

$$\zeta_S^{\kappa} = \begin{cases} \zeta_{S'}^{\kappa} & \kappa \notin \{\hat{\kappa}+\mathbf{e}_k, k=1,\ldots,d\} \\ \zeta_{S'}^{\kappa} + \frac{\kappa_k}{|\kappa|} & \kappa \in \{\hat{\kappa}+\mathbf{e}_k, k=1,\ldots,d\}\ . \end{cases}$$

Considering the fact that $\zeta^{\kappa}_{S'} = 0$ for $\kappa \in \{\hat{\kappa} + \mathbf{e}_k, k = 1, \ldots, d\}$, but $\kappa \notin \partial S'$, we have

$$\zeta^{\kappa}_{S} = \begin{cases} \zeta^{\kappa}_{S'} & \kappa \in (\partial S' \setminus \{\hat{\kappa}\}) \setminus \{\hat{\kappa} + \mathbf{e}_k, k = 1, \ldots, d\} \\ \zeta^{\kappa}_{S'} + \frac{\kappa_k}{|\kappa|} & \kappa \in (\partial S' \setminus \{\hat{\kappa}\}) \cap \{\hat{\kappa} + \mathbf{e}_k, k = 1, \ldots, d\} \\ \frac{\kappa_k}{|\kappa|} & \kappa \in \{\hat{\kappa} + \mathbf{e}_k, k = 1, \ldots, d\} \setminus (\partial S' \setminus \{\hat{\kappa}\}) \ . \end{cases}$$

This allows us to unify the summation over $\partial S' \setminus \{\hat{\kappa}\}$ and the summation over $\{\hat{\kappa} + \mathbf{e}_k, k = 1, \ldots, d\}$ into a single summation, and get

$$f(\mathbf{b}) = \sum_{\kappa \in S} \frac{f^{(\kappa)}(\mathbf{a})}{\kappa!}(\mathbf{b} - \mathbf{a})^{\kappa} + \sum_{\kappa \in \partial S} \zeta^{\kappa}_{S} \int_0^1 \frac{f^{(\kappa)}(\mathbf{x}(t))}{\kappa!}(\mathbf{b} - \mathbf{a})^{\kappa} \, d\, t^{|\kappa|}.$$

This completes the induction proof. □

A special case of this theorem is when the expansion order set S is $\{\kappa \mid |\kappa| < N\}$, in which case its boundary $\partial S = \{\kappa \mid |\kappa| = N + 1\}$. For any $\kappa \in \partial S$, $\kappa - \mathbf{e}_k \in S$ whenever $\kappa_k > 0$, and

$$\zeta^{\kappa}_{S} = \sum_{\substack{k=1 \\ \kappa - \mathbf{e}_k \in S}}^{d} \frac{\kappa_k}{|\kappa|} = 1$$

In this special case, (6.13) becomes the standard Taylor's theorem in multi-dimensional space.

$$f(\mathbf{b}) = \sum_{|\kappa|=0}^{N} \frac{f^{(\kappa)}(\mathbf{a})}{\kappa!}(\mathbf{b} - \mathbf{a})^{\kappa} + \sum_{|\kappa|=N+1} \frac{f^{(\kappa)}(\mathbf{x}_{\kappa})}{\kappa!}(\mathbf{b} - \mathbf{a})^{\kappa}.$$

This shows that Theorem 6.3.3 is indeed an extension to the standard multi-variate Taylor's theorem.

6.3.2 Mathematical formulation

Let $\mathbf{x}_i, i = 1, \ldots, n$ be n distinct value nodes, and $\mathbf{y}_i, i = 1, \ldots, m$ be in m distinct gradient nodes in a d-dimensional space $\mathbb{R}^d$. The target function f is measured at each value node, and the gradient of f is measured at each gradient node. These measurements are given as $\hat{f}(\mathbf{x}_i) \in \mathbb{R}, i = 1, \ldots, n$ and $\nabla\hat{f}(\mathbf{y}_i) \in \mathbb{R}^d, i = 1, \ldots, m$, respectively. The measurement errors are assumed to be mutually independent random variables with zero mean, and their

variances are given as

$$e_{a\,i}^2 = E\left[\left(\hat{f}(\mathbf{x}_i) - f(\mathbf{x}_i)\right)^2\right] \in \mathbb{R}, \quad i = 1, \ldots, n,$$

$$e_{b\,ik}^2 = E\left[\left(\nabla_k \hat{f}(\mathbf{y}_i) - \nabla_k f(\mathbf{y}_i)\right)^2\right] \in \mathbb{R}^d, \quad i = 1, \ldots, m,$$

where ∇_k is the kth component of the gradient operator in $\mathbb{R}^d$. These measurement errors can be set to 0 for exact measurements of the function. The value of our interpolant at point $\mathbf{z} \in \mathbb{R}^d$ is a linear combination of the measurements $\hat{f}(\mathbf{x}_i)$ and $\nabla \hat{f}(\mathbf{x}_i)$:

$$\tilde{f}(\mathbf{z}) = \sum_{i=1}^{n} a_i \hat{f}(\mathbf{x}_i) + \sum_{i=1}^{m} \mathbf{b}_i \cdot \nabla \hat{f}(\mathbf{y}_i), \tag{6.15}$$

where the coefficients $a_i \in \mathbb{R}$ satisfies the normalization condition

$$\sum_{i=1}^{n} a_i = 1, \tag{6.16}$$

and $\cdot$ denotes inner product in $\mathbb{R}^d$.

$$\mathbf{b}_i \cdot \nabla \hat{f}(\mathbf{y}_i) = \sum_{k=1}^{d} b_{ik}\, \nabla_k \hat{f}(\mathbf{y}_i).$$

Let S be a given expansion order set (Definition 6.3.1) of the multi-variate interpolations scheme. The size of S should be approximately equal to or larger than the total order of the given data, $n + m\,d$ in order to take advantage of all the given information about the function f. For example, S can be chosen to cover the least basis (de Boor & Ron, 1992) of the nodes. With any given S, each $f(\mathbf{x}_i)$ can be expanded using the extended Taylor's theorem (Theorem 6.3.3) we proved in the previous subsection.

$$f(\mathbf{x}_i) = \sum_{\kappa \in S} \frac{f^{(\kappa)}(\mathbf{z})}{\kappa!}(\mathbf{x}_i - \mathbf{z})^\kappa + \sum_{\kappa \in \partial S} \eta_S^\kappa \frac{f^{(\kappa)}(\xi_{i\kappa})}{\kappa!}(\mathbf{x}_i - \mathbf{z})^\kappa \ ,$$

where ∂S is the boundary of S (Definition 6.3.2), and

$$\eta_S^\kappa = \sum_{\substack{k=1 \\ \kappa - \mathbf{e}_k \in S}}^{d} \frac{\kappa_k}{|\kappa|}.$$

Define

$$S_k = \{\kappa \mid \kappa + \mathbf{e}_k \in S,\ \kappa_k \geq 0\},$$

which are also expansion order sets by definition. Each $\nabla f(\mathbf{y}_i)$ can also be expanded as

$$\begin{aligned}
\nabla_k f(\mathbf{y}_i) &= \sum_{\kappa \in S_k} \frac{f^{(\kappa+\mathbf{e}_k)}(\mathbf{z})}{\kappa!}(\mathbf{y}_i - \mathbf{z})^\kappa + \sum_{\kappa \in \partial S_k} \eta_{S_k}^\kappa \frac{f^{(\kappa+\mathbf{e}_k)}(\eta_{i\kappa})}{\kappa!}(\mathbf{y}_i - \mathbf{z})^\kappa \\
&= \sum_{\substack{\kappa \in S \\ \kappa_k > 0}} \frac{f^{(\kappa)}(\mathbf{z})}{(\kappa - \mathbf{e}_k)!}(\mathbf{y}_i - \mathbf{z})^{\kappa - \mathbf{e}_k} + \sum_{\substack{\kappa \in \partial S \\ \kappa_k > 0}} \eta_{S_k}^{\kappa - \mathbf{e}_k} \frac{f^{(\kappa)}(\eta_{i\kappa})}{(\kappa - \mathbf{e}_k)!}(\mathbf{y}_i - \mathbf{z})^{\kappa - \mathbf{e}_k}
\end{aligned}$$

where ∂S_k is the boundary of S_k. Incorporate these expansions into (6.15), the residual of the interpolation is

$$\begin{aligned}
r(\mathbf{z}) = \tilde{f}(\mathbf{z}) - f(\mathbf{z}) = &\sum_{\substack{\kappa \in S \\ \kappa \neq \mathbf{0}}} f^{(\kappa)}(\mathbf{z}) \left(\sum_{i=1}^{n} a_i \frac{(\mathbf{x}_i - \mathbf{z})^\kappa}{\kappa!} + \sum_{\substack{k=1 \\ \kappa_k > 0}}^{d} \sum_{i=1}^{m} b_{ik} \frac{(\mathbf{y}_i - \mathbf{z})^{\kappa - \mathbf{e}_k}}{(\kappa - \mathbf{e}_k)!} \right) \\
&+ \sum_{\kappa \in \partial S} \sum_{i=1}^{n} f^{(\kappa)}(\xi_\kappa) \left(a_i \frac{(\mathbf{x}_i - \mathbf{z})^\kappa}{\kappa!} \right) \\
&+ \sum_{\kappa \in \partial S} \sum_{\substack{k=1 \\ \kappa_k > 0}}^{d} \sum_{i=1}^{m} f^{(\kappa)}(\eta_\kappa) \left(b_{ik} \frac{(\mathbf{y}_i - \mathbf{z})^{\kappa - \mathbf{e}_k}}{(\kappa - \mathbf{e}_k)!} \right) \\
&+ \sum_{i=1}^{n} \left(\hat{f}(\mathbf{x}_i) - f(\mathbf{x}_i) \right) a_i + \sum_{k=1}^{d} \sum_{i=1}^{m} \left(\nabla_k \hat{f}(\mathbf{y}_i) - \nabla_k f(\mathbf{y}_i) \right) b_{ik}.
\end{aligned}$$

The $\kappa = \mathbf{0}$ term is canceled out due to the normalization condition (6.16).

Similar to the 1-D interpolation, the "magnitude" $\beta > 0$ and "roughness" $\gamma > 0$ are two given parameters that describe the behavior of the function f. Determination of these parameters is similar to the 1-D case discussed in Sections 6.2.4 and 6.2.5. We consider both parameters as given in this section. We assume that each $f^\kappa(\mathbf{z})$ is a random variable with zero mean and standard deviation $\beta\gamma^{|\kappa|}$. These random variables are assumed to be

mutually independent, and are independent of the measurement errors. The variance of the interpolation residual $r(\mathbf{z})$ under these assumptions is

$$\begin{aligned} E\left[r(\mathbf{z})^2\right] = \beta^2 \sum_{\substack{\kappa\in S\\ \kappa\neq\mathbf{0}}} \gamma^{2|\kappa|} \left(\sum_{i=1}^{n} a_i \frac{(\mathbf{x}_i-\mathbf{z})^\kappa}{\kappa!} + \sum_{\substack{k=1\\ \kappa_k>0}}^{d} \sum_{i=1}^{m} b_{ik} \frac{(\mathbf{y}_i-\mathbf{z})^{\kappa-\mathbf{e}_k}}{(\kappa-\mathbf{e}_k)!} \right)^2 \\ + \beta^2 \sum_{\kappa\in\partial S} \sum_{i=1}^{n} \gamma^{2|\kappa|} \left(a_i \frac{(\mathbf{x}_i-\mathbf{z})^\kappa}{\kappa!} \right)^2 + \beta^2 \sum_{\kappa\in\partial S} \sum_{\substack{k=1\\ \kappa_k>0}}^{d} \sum_{i=1}^{m} \gamma^{2|\kappa|} \left(b_{ik} \frac{(\mathbf{y}_i-\mathbf{z})^{\kappa-\mathbf{e}_k}}{(\kappa-\mathbf{e}_k)!} \right)^2 \\ + \sum_{i=1}^{n} e_{a\,i}^2 a_i^2 + \sum_{k=1}^{d} \sum_{i=1}^{m} e_{b\,ik}^2 b_{ik}^2, \end{aligned}$$

which is in quadratic form of the coefficients vector

$$[a\,b] = [a_1 \dots a_n\, b_{11} \dots b_{m1} \dots\dots b_{1d} \dots b_{md}].$$

In fact, denote the elements in the set S as

$$S = \{\mathbf{0}, \kappa_S^1, \kappa_S^2, \dots, \kappa_S^N\},$$

where each κ_S^l is a d-dimensional index, and N is the size of the set $S \setminus \{\mathbf{0}\}$. Let X^a be an $N \times n$ matrix with each element

$$X_{l\,i}^a = \gamma^{|\kappa_S^l|} \frac{(\mathbf{x}_i-\mathbf{z})^{\kappa_S^l}}{\kappa_S^l!}.$$

Each $X_k^b, k = 1, \dots, d$ be an $N \times m$ matrix with each element

$$X_{k\,l\,i}^b = \begin{cases} 0, & \kappa_{S\,k}^l = 0 \\ \gamma^{|\kappa_S^l|} \dfrac{(\mathbf{x}_i-\mathbf{z})^{\kappa_S^l-\mathbf{e}_k}}{(\kappa_S^l-\mathbf{e}_k)!}, & \kappa_{S\,k}^l > 0 \end{cases}$$

and

$$X^b = \left[X_1^b \dots X_d^b \right]$$

be an $N \times m\,d$ matrix. Also let G be an $n + m\,d \times n + m\,d$ diagonal matrix with

$$G_{jj}^2 = \begin{cases} \displaystyle\sum_{\kappa \in \partial S} \left(\gamma^{|\kappa|} \, \frac{(\mathbf{x}_j - \mathbf{z})^\kappa}{\kappa!} \right)^2, & 1 \le j \le n \\ \displaystyle\sum_{\substack{\kappa \in \partial S \\ \kappa_k > 0}} \left(\gamma^{|\kappa|} \, \frac{(\mathbf{y}_i - z)^{\kappa - \mathbf{e}_k}}{(\kappa - \mathbf{e}_k)!} \right)^2, & j = n + m(k-1) + i, \; 1 \le k \le d, \; 1 \le i \le m \end{cases}$$

and H also be an $n + m\,d \times n + m\,d$ diagonal matrix with

$$H_{jj} = \begin{cases} e_{a\,j}, & j = 1, \ldots, n \\ e_{b\,ik}, & j = n + m(k-1) + i, \quad 1 \le k \le d, \quad 1 \le i \le m \; . \end{cases}$$

The matrix form of the expectation of $r(\mathbf{z})^2$ is the same as in the 1-D case

$$E\left[r(z)^2\right] = [a\,b] \left(\beta^2 \begin{bmatrix} X^{a\,\mathrm{T}} \\ X^{b\,\mathrm{T}} \end{bmatrix} \begin{bmatrix} X^a \, X^b \end{bmatrix} + \beta^2 G^2 + H^2 \right) [a\,b]^{\mathrm{T}}.$$

Denote $n + m\,d \times n + m\,d$ matrix

$$A = \beta^2 \begin{bmatrix} X^{a\,\mathrm{T}} \\ X^{b\,\mathrm{T}} \end{bmatrix} \begin{bmatrix} X^a \, X^b \end{bmatrix} + \beta^2 G^2 + H^2,$$

and length $n + m\,d$ vector

$$c_i = \begin{cases} 1 & i = 1, \ldots, n \\ 0 & i = n+1, \ldots, n\,d + m \; . \end{cases}$$

We get the same quadratic programming problem as in the 1-D case

$$\min \, [a\,b] \, A \, [a\,b]^{\mathrm{T}}, \quad c\,[a\,b]^{\mathrm{T}} = 1. \tag{6.17}$$

Once a and b are calculated by solving this quadratic programming, the value of the interpolant at $\mathbf{z}$ can be computed using (6.15).

6.3.3 Numerical examples

In this section, we apply our multi-variate interpolation scheme to the following example functions in 2-D:

1. A two-dimensional cosine wave $f(x) = \cos(x_1 + x_2)$. The contour plot of this function consists of straight, diagonal lines.
2. The two-dimensional Runge function $f(x) = \dfrac{1}{1 + x_1^2 + x_2^2}$. The contour plot of this function consists of concentric circles centered at the origin.

The domain of interpolation for both functions is $-2 < x_1 < 2, -2 < x_2 < 2$.

Figure 6.14 shows the interpolation results on the two-dimensional cosine function, and Figure 6.15 shows the interpolation results on the two-dimensional Runge function. As can be seen, the interpolation scheme works well in two-dimensional space for arbitrarily scattered datapoints, with and without using gradient information. In addition, the contour lines of the interpolant constructed using gradient information are more accurate than the interpolant constructed without gradient on the same number of grid points. The convergence plots also reveal this information. In both convergence plots, the dotted lines on the upper side of the plots show the L^∞ error of the interpolants constructed without using the gradient information; the dotted lines on the lower side of the plots show the L^∞ error of the interpolants constructed using the gradient information; the solid lines on the upper side of the plots show the L^2 error of the interpolants constructed without using the gradient information; the solid lines on the lower side of the plots show the L^2 error of the interpolants constructed using the gradient information. As seen from the convergence plots, the error of the interpolant constructed with gradient information is much more accurate.

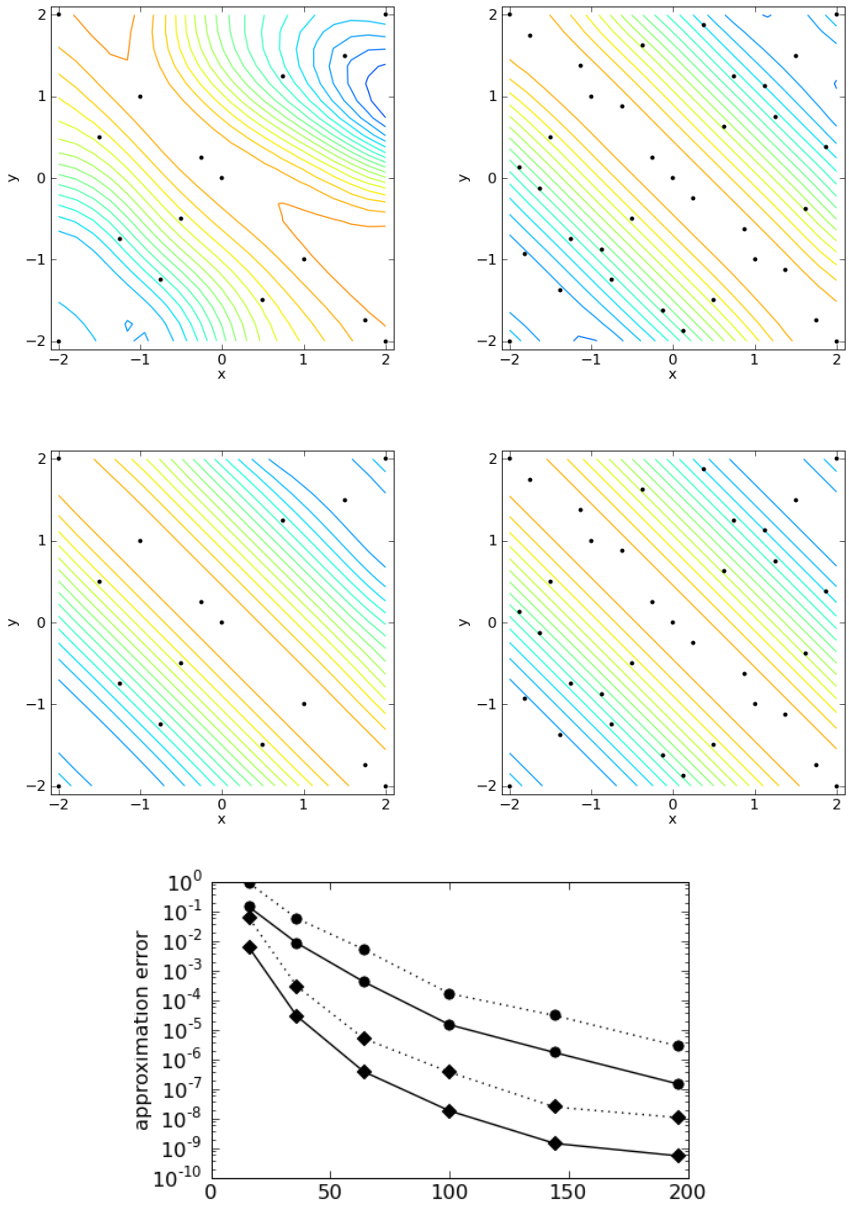

Figure 6.14: Interpolating the 2-D cosine function using 16 (left) and 36 (right) quasi-random grid points. The upper two plots are contour lines of the interpolant constructed without gradient information; the lower two plots are contours of the interpolant constructed with gradient information on each grid point. In the convergence plot (bottom), the horizontal axis is the number of grid points; the solid lines indicate L_2 errors, and the dotted lines indicate L_∞ errors. The upper lines are the approximation errors without using gradient information; the lower lines are the the approximation errors using gradient information.

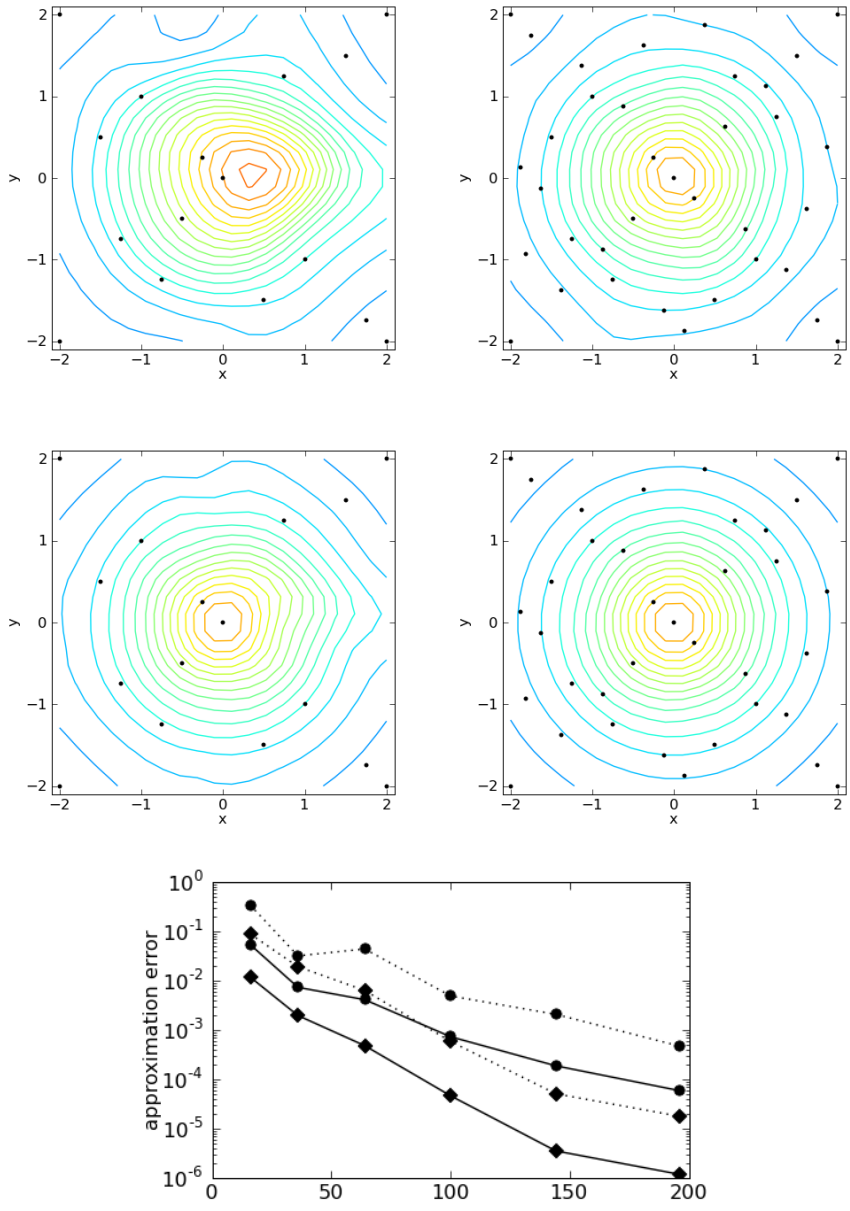

Figure 6.15: Interpolating the 2-D Runge function using 16 (left) and 36 (right) quasi-random grid points. The upper two plots are contour lines of the interpolant constructed without gradient information; the lower two plots are contours of the interpolant constructed with gradient information on each grid point. In the convergence plot (bottom), the horizontal axis is the number of grid points; the solid lines indicate L_2 error, and the dotted lines indicate L_∞ error. The upper lines are the approximation errors without using gradient information; the lower lines are the the approximation errors using gradient information.

Chapter 7

Conclusion and Future Work

The primary goals of this work are to develop an efficient method of solving the adjoint equation for unsteady fluid flow equations, and to use the adjoint solution to efficiently propagate uncertainties in computational fluid dynamics simulations. Both goals have been achieved for incompressible Navier-Stokes flow simulations in complex geometry using unstructured grids. Two ways of efficiently solving the adjoint equation for time-dependent problems have been demonstrated: the Monte Carlo forward-time adjoint solver, and the dynamic checkpointing method. We applied the dynamic checkpointing method in an unstructured incompressible Navier-Stokes solver, and integrated it into CDP, the incompressible flow solver used at the Center for Turbulence Research at Stanford University. We also developed two approaches for using the adjoint solution in propagating uncertainties. One approach is using the adjoint sensitivity gradient to accelerate Monte Carlo methods in quantification of risk. The other approach is a multi-variate interpolation scheme. The scheme uses the gradient information on grid points to improve accuracy. It works for arbitrary scattered grid points, and converges exponentially to the objective function.

Two important areas of future work are improvements to the Monte Carlo adjoint solver and applications of the interpolation scheme. To further improve the accuracy of the Monte Carlo adjoint solver and reduce its cost, variance-reduction methods can be used on the Monte Carlo linear solver. In addition, formulating the linear system on a different linear basis may significantly reduce the variance. For example, we plan to formulate the adjoint equation on a Fourier basis and analyze its impact on the accuracy of the Monte Carlo solution. If we can reduce the variance of our Monte Carlo adjoint solver, we can apply this method to the adjoint of the Navier-Stokes equations.

The second area of future work is in the application of the multi-variate interpolation method. We plan to apply this method to the Navier-Stokes equations, and compare it to uncertainty quantification methods such as polynomial chaos and collocation methods. In addition, we want to use the interpolation method to augment collocation methods. In a collocation method, it is possible that one level of tensor or sparse grid is not sufficient to obtain accurate results, while the next grid level is too expensive to calculate. In this case, one can add as many grid points as are affordable, and use our interpolation method to obtain the value at the uncalculated points in the next level grid. Such a method could make collocation methods more practical when the dimensionality of the random space is high.

Appendix A
The discrete adjoint scheme

The scheme for solving the unsteady incompressible Navier-Stokes equation in the CDP flow solver is

$$
\begin{aligned}
u_m^{k*} &= 2u_m^{k-1} - u_m^{k-2} \\
u^{k-\frac{2}{3}} &= \frac{4}{3}u^{k-1} - \frac{1}{3}u^{k-2} \\
\frac{3}{2}\frac{u^{k*} - u^{k-\frac{2}{3}}}{\Delta t} &= -u_m^{k*} \cdot \nabla u^{k*} + \nabla \cdot \mu \nabla u^{k*} - \nabla p^{k-1} \\
u^{k-} &= u^{k*} + \frac{2\Delta t}{3}\nabla p^{k-1} \\
\Delta p^k &= \frac{3}{2\Delta t}\nabla \cdot u^{k-} \\
u^k &= u^{k-} - \frac{2\Delta t}{3}\nabla p^k \\
u_m^k &= P_F\, u^{k-} - \frac{2\Delta t}{3}\nabla_F\, p^k \ .
\end{aligned}
\tag{A.1}
$$

We have five state variables in the scheme: $u^k, u^{k-1}, u_m^k, u_m^{k-1}, p^k$. Δt is the size of the time step. The convection operator $u_m \cdot \nabla$, the gradient operators ∇, the divergence operator $\nabla\cdot$, the Laplace operator Δ, the facial gradient operator ∇_F, and the facial projection operator P_F are all in the Simplex-Superposition formulation (Ham, 2008).

The tangent linear scheme corresponding to the Navier-Stokes scheme is

$$
\begin{aligned}
&\delta u_m^{k*} = 2\delta u_m^{k-1} - \delta u_m^{k-2} \\
&\delta u^{k-\frac{2}{3}} = \frac{4}{3}\delta u^{k-1} - \frac{1}{3}\delta u^{k-2} \\
&\frac{3}{2}\frac{\delta u^{k*} - \delta u^{k-\frac{2}{3}}}{\Delta t} = -u_m^{k*}\cdot\nabla\delta u^{k*} - \delta u_m^{k*}\cdot\nabla u^{k*} + \nabla\cdot\mu\nabla\delta u^{k*} - \nabla\delta p^{k-1} \\
&\delta u^{k-} = \delta u^{k*} + \frac{2\Delta t}{3}\nabla\delta p^{k-1} \\
&\Delta\delta p^{k} = \frac{3}{2\Delta t}\nabla\cdot\delta u^{k-} \\
&\delta u^{k} = \delta u^{k-} - \frac{2\Delta t}{3}\nabla\delta p^{k} \\
&\delta u_m^{k} = P_F\, u^{k-} - \frac{2\Delta t}{3}\nabla_F\,\delta p^{k}\ ,
\end{aligned}
\tag{A.2}
$$

where $\delta u^k, \delta u^{k-1}, \delta u_m^k, \delta u_m^{k-1}, \delta p^k$ are the tangent linear state variables. The only difference from the Navier-Stokes scheme is in the scheme is the convection operator, which is linearized into two convection operators.

With this tangent linear scheme, we derive the adjoint scheme of this tangent linear scheme

$$
\begin{aligned}
&\Delta q^k = \frac{\hat{p}^k + \nabla\cdot\hat{u}^k}{\Delta t} + \nabla_F\cdot\hat{u}_m^{k*} \\
&\hat{u}^{k-} = \hat{u}^k + \Delta t\left(P_N\,\hat{u}_m^{k*} - \nabla q^k\right) \\
&\frac{3}{2}\frac{\hat{u}^{k-\frac{2}{3}} - \hat{u}^{k-}}{\Delta t} = u_m^{k*}\cdot\nabla\hat{u}^{k-\frac{2}{3}} + \nabla\cdot\mu\nabla\hat{u}^{k-\frac{2}{3}} \\
&\hat{u}^{k-1} = \frac{4}{3}\hat{u}^{k-\frac{2}{3}} - \frac{1}{3}\hat{u}^{k+\frac{1}{3}} \\
&\hat{u}_m^{k} = -\frac{2}{3}\,\nabla u^{k*}\cdot\hat{u}^{k-\frac{2}{3}} \\
&\hat{p}^{k-1} = \nabla\cdot\left(\hat{u}^{k-\frac{2}{3}} - \hat{u}^{k-}\right) \\
&\hat{u}_m^{k-1*} = 2\hat{u}_m^{k} - \hat{u}_m^{k+1}\ ,
\end{aligned}
\tag{A.3}
$$

where $\hat{u}^k, \hat{u}^{k-1}, \hat{u}_m^k, \hat{u}_m^{k-1}, \hat{p}^k$ are the adjoint state variables. The Laplace operator Δ is self-adjoint, so it is the adjoint of the Laplace operator in the tangent linear scheme. The divergence operator $\nabla\cdot$ is the adjoint operator of the gradient operator ∇ in the tangent linear scheme, while the gradient operator ∇ is the adjoint operator of the divergence

operator. The facial to nodal divergence operator $\nabla_F \cdot$ is the adjoint operator of the facial gradient operator ∇_F. The facial to nodal projection P_N is the adjoint operator to the nodal to facial projection P_F. The convection operator $u_m \cdot \nabla$ is adjoint to the negative of itself.

This discrete adjoint scheme (A.3) and the tangent linear Navier-Stokes schemes (A.2) satisfy the discrete adjoint relation:

$$\begin{aligned} &\iiint \delta u^k \cdot \hat{u}^k - \frac{1}{3}\,\delta u^{k-1} \cdot \hat{u}^{k+\frac{1}{3}} + \\ &\qquad \Delta t \left(\delta u_m^k \cdot \hat{u}_m^{k*} - \delta u_m^{k-1} \cdot \hat{u}_m^{k+1} + \frac{2}{3}\,\delta p^k \hat{p}^k \right) dx \\ = &\iiint \delta u^{k-1} \cdot \hat{u}^{k-1} - \frac{1}{3}\,\delta u^{k-2} \cdot \hat{u}^{k-\frac{2}{3}} + \\ &\qquad \Delta t \left(\delta u_m^{k-1} \cdot \hat{u}_m^{k-1*} - \delta u_m^{k-2} \cdot \hat{u}_m^k + \frac{2}{3}\,\delta p^{k-1} \hat{p}^{k-1} \right) dx \,. \end{aligned} \tag{A.4}$$

The following algebraic manipulations of the discrete adjoint scheme (A.3) and the tangent

linear Navier-Stokes schemes (A.2) proves this discrete adjoint relation.

$$
\begin{aligned}
&\iiint \delta u^k \cdot \hat{u}^k - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}} + \Delta t\left(\delta u_m^k\cdot\hat{u}_m^{k*} - \delta u_m^{k-1}\cdot\hat{u}_m^{k+1} + \frac{2}{3}\,\delta p^k\hat{p}^k\right)dx\\
&= \iiint\left(\delta u^{k-} - \frac{2\Delta t}{3}\nabla\delta p^k\right)\cdot\hat{u}^k + \Delta t\left(P_F\,u^{k-} - \frac{2\Delta t}{3}\nabla_F\,\delta p^k\right)\cdot\hat{u}_m^{k*} + \frac{2\Delta t}{3}\,\delta p^k\hat{p}^k\\
&\qquad - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}} - \Delta t\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}dx\\
&= \iiint \delta u^{k-}\cdot\left(\hat{u}^k + \Delta t\,P_N\,\hat{u}_m^{k*}\right) + \frac{2\Delta t}{3}\,\delta p^k\left(\nabla\cdot\hat{u}^k + \Delta t\nabla_F\cdot\hat{u}_m^{k*} + \hat{p}^k\right)\\
&\qquad - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}} - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx\\
&= \iiint \delta u^{k-}\cdot\left(\hat{u}^k + \Delta t\,P_N\,\hat{u}_m^{k*}\right) + \frac{2\Delta t^2}{3}\,\delta p^k\,\Delta q^k - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}}\\
&\qquad - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx\\
&= \iiint \delta u^{k-}\cdot\left(\hat{u}^k + \Delta t\,P_N\,\hat{u}_m^{k*}\right) + \frac{2\Delta t^2}{3}\,\Delta\delta p^k\,q^k - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}}\\
&\qquad - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx\\
&= \iiint \delta u^{k-}\cdot\left(\hat{u}^k + \Delta t\,P_N\,\hat{u}_m^{k*}\right) + \Delta t\,\nabla\cdot\delta u^{k-}\,q^k - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}}\\
&\qquad - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx\\
&= \iiint \delta u^{k-}\cdot\left(\hat{u}^k + \Delta t\,P_N\,\hat{u}_m^{k*} - \Delta t\,\nabla q^k\right) - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}} - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx\\
&= \iiint\left(\delta u^{k*} + \frac{2\Delta t}{3}\nabla\delta p^{k-1}\right)\cdot\hat{u}^{k-} - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}} - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx\\
&= \iiint \delta u^{k*}\cdot\left(\hat{u}^{k-\frac{2}{3}} - \frac{2\Delta t}{3}\left(u_m^{k*}\cdot\nabla\hat{u}^{k-\frac{2}{3}} + \nabla\cdot\mu\nabla\hat{u}^{k-\frac{2}{3}}\right)\right) - \frac{2\Delta t}{3}\,\delta p^{k-1}\nabla\cdot\hat{u}^{k-}\\
&\qquad - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}} - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx\\
&= \iiint\left(\delta u^{k*} + \frac{2\Delta t}{3}\left(u_m^{k*}\cdot\nabla\delta u^{k*} - \nabla\cdot\mu\nabla\delta u^{k*}\right)\right)\cdot\hat{u}^{k-\frac{2}{3}} - \frac{2\Delta t}{3}\,\delta p^{k-1}\nabla\cdot\hat{u}^{k-}\\
&\qquad - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}} - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx\\
&= \iiint\left(\delta u^{k-\frac{2}{3}} - \frac{2\Delta t}{3}\left(\delta u_m^{k*}\cdot\nabla u^{k*} + \nabla\delta p^{k-1}\right)\right)\cdot\hat{u}^{k-\frac{2}{3}} - \frac{2\Delta t}{3}\,\delta p^{k-1}\nabla\cdot\hat{u}^{k-}\\
&\qquad - \frac{1}{3}\,\delta u^{k-1}\cdot\hat{u}^{k+\frac{1}{3}} - \Delta t\,\delta u_m^{k-1}\cdot\hat{u}_m^{k+1}\,dx
\end{aligned}
$$

$$
\begin{aligned}
&= \iiint \delta u^{k-\frac{2}{3}} \cdot \hat{u}^{k-\frac{2}{3}} + \Delta t\, \delta u_m^{k*} \cdot \hat{u}_m^k + \frac{2\Delta t}{3}\, \delta p^{k-1} \nabla \cdot \hat{u}^{k-\frac{2}{3}} - \frac{2\Delta t}{3}\, \delta p^{k-1} \nabla \cdot \hat{u}^{k-} \\
&\qquad - \frac{1}{3}\, \delta u^{k-1} \cdot \hat{u}^{k+\frac{1}{3}} - \Delta t\, \delta u_m^{k-1} \cdot \hat{u}_m^{k+1}\, dx \\
&= \iiint \left(\frac{4}{3} \delta u^{k-1} - \frac{1}{3} u^{k-2} \right) \cdot \hat{u}^{k-\frac{2}{3}} + \Delta t\, \left(2\delta u_m^{k-1} - \delta u_m^{k-2} \right) \cdot \hat{u}_m^k \\
&\qquad + \frac{2\Delta t}{3}\, \delta p^{k-1} \nabla \cdot \left(\hat{u}^{k-\frac{2}{3}} - \hat{u}^{k-} \right) - \frac{1}{3}\, \delta u^{k-1} \cdot \hat{u}^{k+\frac{1}{3}} - \Delta t\, \delta u_m^{k-1} \cdot \hat{u}_m^{k+1}\, dx \\
&= \iiint \delta u^{k-1} \cdot \left(\frac{4}{3} \hat{u}^{k-\frac{2}{3}} - \frac{1}{3} \hat{u}^{k+\frac{1}{3}} \right) - \frac{1}{3}\, u^{k-2} \cdot \hat{u}^{k-\frac{2}{3}} + \Delta t\, u_m^{k-1} \cdot \left(2\hat{u}_m^k - \hat{u}_m^{k+1} \right) \\
&\qquad - \Delta t\, \delta u_m^{k-2} \cdot \hat{u}_m^k + \frac{2\Delta t}{3}\, \delta p^{k-1} \hat{p}^{k-1}\, dx \\
&= \iiint \delta u^{k-1} \cdot \hat{u}^{k-1} - \frac{1}{3}\, \delta u^{k-2} \cdot \hat{u}^{k-\frac{2}{3}} \\
&\qquad + \Delta t \left(\delta u_m^{k-1} \cdot \hat{u}_m^{k-1*} - \delta u_m^{k-2} \cdot \hat{u}_m^k + \frac{2}{3}\, \delta p^{k-1} \hat{p}^{k-1} \right) dx\, .
\end{aligned}
$$

Bibliography

Abergel, F. & Temam, R. 1990 On some control problems in fluid mechanics. *Theoretical and Computational Fluid Dynamics* **1** (6), 303–325.

Alexandrov, V. N. 1998 Efficient parallel Monte Carlo methods for matrix computations. *Mathematics and Computers in Simulation* **47**, 113–122.

Babuska, I., Nobile, F. & Tempone, R. 2007 A stochastic collocation method for elliptic partial differential equations with random input data. *SIAM J. Numer. Anal.* **45** (3), 1005–1034.

Bae, H., Grandhi, R. V. & Canfield, R. A. 2004 An approximation approach for uncertainty quantification using evidence theory. *Reliability engineering and systems safety* **86** (3), 215–225.

Bewley, T. R. 2001 Flow control: new challenges for a new renaissance. *Progress in Aerospace Sciences* **37** (1), 21–58.

Bewley, T. R., Moin, P. & Temam, R. 2001 DNS-based predictive control of turbulence: an optimal target for feedback algorithms. *J. Fluid Mech.* **447**, 179–225.

de Boor, C. & Ron, A. 1992 The least solution for the polynomial interpolation problem. *Mathematische Zeitschrift* **210** (1), 347–378.

Bose, D. & Wright, M. 2006 Uncertainty analysis of laminar aeroheating predictions for mars entries. In *38th AIAA Thermophysics Conference* (ed. V. Alexandrov et al.), pp. 652–662. AIAA 2005-4682.

Boyd, J. 1999 A numerical comparison of seven grids for polynomial interpolation on the interval. *Computers and Mathematics with Applications* **38**, 35–50.

Bryson, A. E., Desai, M. N. & Hoffman, W. C. 1969 Energy state approximation in performance optimization of supersonic aircraft. *Journal of Aircraft* **6** (12), 488–490.

Charpentier, I. 2001 Checkpointing schemes for adjoint codes: Application to the meteorological model Meso-NH. *SIAM J. Sci. Comput.* **22** (6), 2135–2151.

Cheng, H. & Sandu, A. 2007 Numerical study of uncertainty quantification techniques for implicit stiff systems. In *Proceedings of the 45th annual southeast regional conference*, pp. 367 – 372.

Choi, B., Temam, R., Moin, P. & Kim, J. 1993 Feedback control for unsteady flow and its application to the stochastic burgers equation. *J. Fluid Mech* **253**, 509–543.

Courtier, P., Derber, J., Errico, R., Louis, J. F. & Vukicevic, T. 2002 Important literature on the use of adjoint, variational methods and the Kalman filter in meteorology. *Tellus A* **45** (5), 342–357.

Debusschere, B. J., Najm, H. N., Pbay, P. P., Knio, O. M., Ghanem, R. G. & Maitre, O. P. Le 2005 Numerical challenges in the use of polynomial chaos representations for stochastic processes. *SIAM J. Sci. Comput.* **26** (2), 698–719.

Dimov, I. 1991 Minimization of the probable error for some Monte Carlo methods. In *Proceedings of the Summer School on Mathematical Modelling and Scientific Computations*, pp. 159–170. Alberta, Bulgaria: Publication House of the Bulgarian Acad. Sci.

Dimov, I. 1998 Monte Carlo algorithms for linear problems. In *Lecture Notes of the 9th International Summer School on Probability Theory and Mathematical Statistics* (ed. N. M. Yanev), pp. 51–71. SCT Publishing.

Dimov, I. & Tonev, O. 1993 Random walk on distant mesh points Monte Carlo methods. *Journal of Statistical Physics* **70**, 1333–1342.

Draper, D. 1995 Assessment and propagation of model uncertainty. *Journal of the Royal Statistical Society. Series B* **57** (1), 45–97.

Eldred, M. S., Giunta, A. A., Waanders, B. G. Van Bloemen, Jr., S. F. Wojtkiewics, Hart, W. E. & Alleva, M. 2002 Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis version 3.0 developers manual. Technical Report, Sandia National Labs., Albuquerque, NM and Livermore, CA.

Ellison, S. L. R., Rosslein, M. & Williams, A. 2000 *Quantifying uncertainty in analytical measurement*. CEPIS.

Fey, U., Konig, M. & Eckelmann, H. 1998 A new strouhal–reynolds-number relationship for the circular cylinder in the range $47 \leq re \leq 2 \times 10^5$. *Physics of Fluids* **10** (7), 1547–1549.

Forsythe, S. E. & Liebler, R. A. 1950 *Mathematical Tables and Other Aids to Computation*, chap. Matrix Inversion by a Monte Carlo Method, pp. 127–129.

Ghanem, R. & Spanos, P. D. 1990 Polynomial chaos in stochastic finite elements. *Journal of Applied Mechanics* **57** (1), 197–202.

Glynn, P. W. & Iglehart, D. L. 1989 Importance sampling for stochastic simulations. *Manage. Sci.* **35** (11), 1367–1392.

Glynn, P. W. & Szechtman, R. 2002 Some new perspectives on the method of control variates. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pp. 27–49. Springer-Verlag.

Gordon, W. & Wixom, J. 1978 Shepard's method of "metric interpolation" to bivariate and multivariate interpolation. *Math. Comp.* **32** (141), 253–264.

Green, L., Lin, H. & Khalessi, M. 2002 Probabilistic methods for uncertainty propagation applied to aircraft design. In *20th AIAA Applied Aerodynamics Conference*.

Griewank, A. 1992 Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software* **1**, 35–54.

Griewank, A. 2003 A mathematical view of automatic differentiation. *Acta Numerica* pp. 321–398.

Griewank, A. & Walther, A. 2000 Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.* **26** (1), 19–45.

Ham, F. 2008 Improved scalar transport for unstructured finite volume methods using grids based on simplex superposition. In *Annual Research Briefs 2008*. Center for Turbulence Research, Stanford University, NASA Ames.

Ham, F., Mattsson, K., Iaccarino, G. & Moin, P. 2007 *Towards Time-Stable and Accurate LES on Unstructured Grids. Complex Effects in Large Eddy Simulation*, , vol. 56. Springer.

Hammersley, J. M. & Handscomb, D. C. 1964 *Monte Carlo methods*. Methuen.

Helton, J. C. & Oberkampf, W. L. 2004 Alternative representations of epistemic uncertainty. *Reliability Engineering and System safety* **85** (1), 1–10.

Heuveline, V. & Walther, A. 2006 Online checkpointing for parallel adjoint computation in pdes: Application to goal oriented adaptivity and flow control. In *Proceedings of Euro-Par 2006 Parallel Processing* (ed. W. Nagel et al.), pp. 689–699. Springer.

Hinze, M. & Sternberg, J. 2005 A-revolve: An adaptive memory- and run-time-reduced procedure for calculating adjoints; with an application to the instationary navier-stokes system. *Optimization Methods and Software* **20**, 645–663.

Hosder, S., Perez, R. & Walters, R. 2006 A non-intrusive polynomial chaos method for uncertainty propagation in cfd simulations. In *44th AIAA Aerospace Sciences Meeting and Exhibit*.

Hosea, M. E. & Shampine, L. F. 1996 Analysis and implementation of tr-bdf2. *Appl. Numer. Math.* **20** (1-2), 21–37.

Isukapalli, S. S., Roy, A. & Georgopoulos, P. G. 1998 Stochastic response surface methods (srsms) for uncertainty propagation: Application to environmental and biological systems. *Risk Analysis* **18** (3), 351–363.

Jameson, A. 1988 Aerodynamic design via control theory. *J. of Scientific Computing* **3**, 233–260.

Jameson, A. 2003 Time integration methods in computational aerodynamics. In *AFOSR Work Shop on Advances and Challenges in Time-Integration of PDEs*. Providence, RI.

Kowarz, A. & Walther, A. 2006 Optimal checkpointing for time-stepping procedures. In *Proceedings of ICCS 2006, LNCS 3994* (ed. V. Alexandrov et al.), pp. 541–549. Springer.

Kuczera, G. & Parent, E. 1998 Monte Carlo assessment of parameter uncertainty in conceptual catchment models: the metropolis algorithm. *Journal of Hydrology* **211**, 69–85.

Mathelin, L., Hussaini, M. & Zang, T. 2005 Stochastic approaches to uncertainty quantification in cfd simulations. *Numerical Algorithms* **38-1**, 209–236.

Metropolisand, N. & Ulam, S. 1949 The Monte Carlo method. *Journal of the American Statistical Association* **44**, 335–341.

Morinishi, Y., Lund, T. S., Vasilyev, O. V. & Moin, P. 1998 *Journal of computational physics* **143** (1), 90–124.

Mourelatos, Z. P. & Zhou, J. 2004 Uncertainty quantification using evidence theory in multidisciplinary design optimization. *Reliability Engineering and System Safety* **85** (1), 281–294.

Niederreiter, H. 1992 *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Okten, G. 2005 Solving linear equations by Monte Carlo simulation. *SIAM J. Sci. Comput.* **27** (2), 511–531.

Palmer, T. N. 2000 Predicting uncertainty in forecasts of weather and climate. *Rep. Prog. Phys.* **63**, 71–116.

Pantano, C. 2007 Least-squares dynamic approximation method for evolution of uncertainty in initial conditions of dynamical systems. *Physical Review E* **76**.

Pharr, M. & Hanrahan, P. 2000 Monte Carlo evaluation of non-linear scattering equations for subsurface reflection. *Proceedings of SIGGRAPH* .

Putko, M. M., Taylor, A. C., Newman, P. A. & Green, L. L. 2001 Approach for input uncertainty propagation and robust design in cfd using sensitivity derivatives. In *15th AIAA Computational Fluid Dynamics Conference*.

Reuther, J. J., Jameson, A., Alonso, J. J., Rimlinger, M. J. & Saunders, D. 1999 Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers. *Journal of aircraft* **36** (1), 51–60.

Roache, P. J. 1997 Quantification of uncertainty in computational fluid dynamics. *Annual Review of Fluid Mechanics* **29**, 123–160.

Roulston, M. S., Bolton, G. E., Kleit, A. N. & Sears-Collins, A. L. 2006 A laboratory study of the benefits of including uncertainty information in weather forecasts. *Weather and Forecasting* **21**, 116.

Rubinstein, O. & Choudhari, M. 2005 Uncertainty quantification for systems with random initial conditions using wiener–hermite expansions. *Studies in Applied Mathematics* **114** (2), 167–188.

Rubinstein, R. Y. 1981 *Simulation and the Monte Carlo Method*. John Wiley and Sons, New York.

Runge, C. 1901 Uber empirische funktionen und die interpolation zwischen aquidistanten ordinaten. *Zeitschrift fur Mathematik und Physik* **46**, 224–243.

Sobol, I. M. 1973 *Monte Carlo Numerical Methods*, in russian edn. Moscow, Russian: Nauka.

Srinivasan, A. 2003 Improved Monte Carlo linear solvers through non-diagonal splitting. *ICCSA (3)* pp. 168–177.

Stumm, P. & Walther, A. 2008 Towards the economical computation of adjoints in pdes using optimal online checkpointing. Deutsche Forschungsgemeinschaft Schwerpunktprogramm 1253, Tech. Rep. SPP 1253-15-04.

Tan, C. J. K. 2002 Solving systems of linear equations with relaxed Monte Carlo method. *The Journal of Supercomputing* **22**, 113–123.

Tan, C. J. K. & Alexandrov, V. 2001 Relaxed Monte Carlo linear solver. In *Computational Science (Part II)* (ed. V. N. Alexandrov, J. J. Dongarra, B. A. Juliano, R. S. Renner & C. J. K. Tan), *Lecture Notes in Computer Science*, vol. 2074, pp. 1289–1298. Springer-Verlag.

Tokumaru, P. T. & Dimotakis, P. E. 2006 Rotary oscillation control of a cylinder wake. *Journal of Fluid Mechanics* **224**, 77–90.

Trucano, T. G. 1998 Prediction and uncertainty in computational modeling of complex phenomena: A whitepaper.

Walther, A. & Griewank, A. 2004 Advantages of binomial checkpointing for memory-reduced adjoint calculations. *Numerical Mathematics and Advanced Applications* pp. 834–843.

Westlake, J. R. 1968 *A Handbook of Numerical Matrix Inversion and Solution of Linear Equations*. John Wiley and Sons, New York.

Wojtkiewicz, S. F., Eldred, M. S., Field, R. V., Urbina, A. & Red-Horse, J. R. 2001 Uncertainty quantification in large computational engineering models. In *42nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials (SDM) Conference*.

Xiu, D. & Karniadakis, G. E. 2002 The wiener–askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing* **24** (2), 619–644.

Xiu, D. & Karniadakis, G. E. 2003 Modeling uncertainty in flow simulations via generalized polynomial chaos. *Journal of Computational Physics* **187** (1), 137–167.

Yu, Y., Zhao, M., Lee, T., Pestieau, N., Bo, W., Glimm, J. & Grove, J. W. 2006 Uncertainty quantification for chaotic computational fluid dynamics. *J. Comput. Phys.* **217** (1), 200–216.

Zimmermann, H. 1996 *Fuzzy set theory–and its applications*, 3rd edn. Kluwer Academic Publishers.