Efficient Computational Methods for Uncertainty Quantification of Large Systems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Gaurav Gaurav

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Prof. Steven F. Wojtkiewicz

May, 2011

Acknowledgements

Following a famous Indian adage

गुरु गोविन्द दोउ खड़े, कीके लागू पाय | बलिहारी गुरु आपने, गोविन्द दियो बताय ||	*I met my teacher (guru) and God standing together and was confused whom do I pay respect first. God told me that guru is always greater than Him because he has shown the path of knowledge.*

I would like to begin by expressing my deepest gratitude to my advisor Professor Steven Wojtkiewicz for being an excellent teacher and mentor to me. His knowledge, vision, flexibility and patience have inspired me to work hard. The freedom which he let me enjoy during my Ph.D. research allowed me to work at my optimal potential. Prof. Wojtkiewicz, thank you for painstakingly reviewing all the not-so-professional text I wrote; it has been a great honor and privilege working with you.

I would like to thank Professors Henryk Stolarki, Randal Barnes and Yousef Saad for serving on my Ph.D. committee and providing constructive criticism of my research and dissertation. I am thankful to Professor Barnes for being my mentor in the preparing for future faculty courses. I am grateful to him for the fruitful discussions that we had and his candid comments. The genius of Professors Stolarski and Saad has always motivated me to keep working hard.

I would like to acknowledge all my teachers from the Civil Engineering, Electrical Engineering, and Computer Science departments who played a vital role in my academic and intellectual growth.

I am thankful to the Civil Engineering department for providing me financial support through assistantships and the Burgum/Sommerfeld fellowships. I also acknowledge the Graduate School of the University of Minnesota for awarding me the doctoral

dissertation fellowship in the final year of my Ph.D. research.

Many thanks to all my friends who gave me a fun social life during my stay at the University of Minnesota despite their occasional 'humorous remarks' regarding my frequent absence from Friday night parties and weekend outings. I will miss the outings I did attend, especially all the hiking and canoeing trips.

Finally, my warmest thanks to my parents and my sister who always kept me motivated. They always encouraged me to be independent and pursue my dreams to the extent I wanted. This dissertation would not have been possible without their continuous support, occasional hardships to keep me on track, understanding and endless love.

Dedication

To my parents and sister.

Abstract

The quest to design environment-friendly and sustainable engineering systems has witnessed more and more fervent efforts in recent years. With the growth of affordable large-capacity computing resources, predictive, science-based computational models have become instrumental in this pursuit. The present work develops efficient computational methods for the uncertainty analysis of large dynamical and mechanical systems with local nonlinearities and uncertainties. Two approaches have been utilized: (i) reduction of the size of the system, and (ii) use of parallel computing resources.

The first approach utilizes the response of a nominal system to efficiently compute the response of related systems; three types of analysis methods have been developed. The first method can be utilized for efficient modal analysis of undamped linear systems with local stiffness uncertainties. The second method can perform efficient frequency domain analysis of linear systems with local damping and stiffness uncertainties. The third method can be utilized for efficient time domain analysis of systems with local nonlinearities and uncertainties. These methods provide gains in computational efficiency approaching three orders of magnitude for moderately-sized computational models compared to the corresponding conventional methods. The gains in computational efficiency are expected to be more pronounced as the dimensionality of the system is increased.

The second approach to increase computational efficiency utilizes modern parallel computing devices, specifically, graphics processing units (GPUs) to perform uncertainty analysis of computational models. A variety of uncertainty quantification methods have been implemented on a GPU. The gains in computational efficiency compared to corresponding CPU-based implementations range from one to three orders of magnitude. These GPU implementations are expected to serve as initial bases for further developments in the use of GPUs in this field.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Most engineering systems can be modeled by a system of partial differential equations (PDEs), ordinary differential equations (ODEs), integral equations (IEs), algebraic equations (AEs), or a combination of these resulting from the application of the basic principles of physics, such as the D'Alembert's principle, Hamilton's principle, and the principle of virtual work. The resulting system of equations are in general, nonlinear. In addition to the nonlinearities present, accurate characterization of the system's behavior is further complicated by the presence of uncertainty in the system's parameters and the inputs to the system. This uncertainty can arise from two sources [1]. The first source stems from the impossibility of obtaining a complete deterministic description of a physical phenomenon, which is also the basis of the Uncertainty Principle [2] in quantum mechanics. The second source results from our incomplete knowledge of the phenomenon being observed. Much of the modern literature terms these two types of uncertainties as aleatoric and epistemic, respectively.

Due to these inherent nonlinearities and uncertainties, it is extremely difficult if not impossible to obtain closed-form analytical solutions of the system of equations formulated to model a given physical system. Therefore, in order to predict the future behavior of these systems, science-based computational models have become instrumental. The accuracy of the computational models greatly depends on (i) the resolution of the discretized state-space of the system in question and (ii) the accuracy of the identification and characterization of the system parameters. In general, these computational models provide increasingly accurate descriptions of the underlying

physical system as the resolution of the system's discretized state-space is increased. However, in doing so, the size of the system's model can increase many-fold rendering the analysis procedure computationally very expensive. The methods employed for the identification and characterization of the system's parameters are usually iterative in nature and require multiple analyses of the system's model subjected to different sets of parameters and/or loading conditions which further adds to the computational demand of prediction models. Such methods are collectively studied under the aegis of uncertainty quantification. The field of uncertainty quantification encompasses the following types of analyses:

- uncertainty analysis
- sensitivity analysis
- model validation and calibration
- design exploration and optimization

Figure 1.1 depicts the functioning of these methods. One can observe the common theme

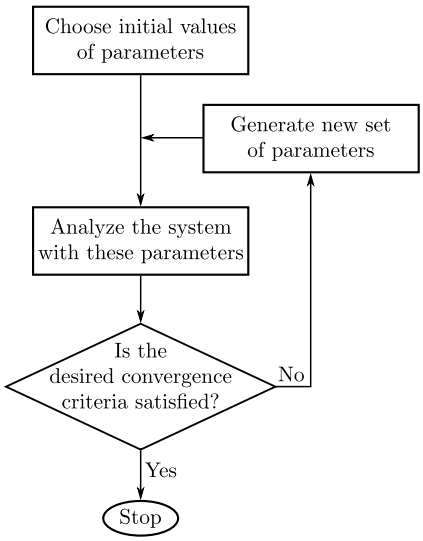


Figure 1.1: General framework of uncertainty quantification methods

of reanalysis among these methods. That is, it is required to analyze nominally similar

systems with slightly modified parameters. When combined with high-performance computing, these uncertainty quantification methods enable researchers to analyze and predict the behavior of physical systems in a concrete and timely manner. The present work develops efficient computational methods for the uncertainty analysis of large dynamical and mechanical systems with local features, such as nonlinearities and uncertainties. The developed methods are based on two types of approaches:

- Use of mathematical techniques to reduce the size of the computational model of the system in question
- Use of contemporary parallel computing resources to render the associated computations more efficient

All engineering systems can be conceptualized as an input/output model, as shown in Figure 1.2. Here, $\mathcal{S}(\mathbf{p}_u, \mathbf{p}_n)$ represents the mathematical model of the system, $\mathbf{p}_u$

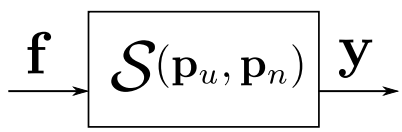


Figure 1.2: Conceptual input/output model

and $\mathbf{p}_n$ denote the system parameters that are involved respectively in the uncertain and nonlinear relationships, $\mathbf{f}$ is the input(s) to the system, and $\mathbf{y}$ is the output(s) of interest.

For example, in structural mechanics analyses, the nonlinear relationships denoted by $\mathbf{p}_n$ in $\mathcal{S}(\mathbf{p}_u, \mathbf{p}_n)$, usually arise from either geometrical or material nonlinearities. Geometrical nonlinearities include large-amplitude deflections and deadbands in joints and control systems while material nonlinearities include nonlinear stress-strain relationships, *i.e.*, nonlinear stiffness characteristics and hysteresis loops, which affect the damping characteristics of the system. The uncertainty, denoted by $\mathbf{p}_u$ in $\mathcal{S}(\mathbf{p}_u, \mathbf{p}_n)$, can manifest itself in the form of randomness in material properties, boundary conditions and/or initial conditions of the system. In addition, the input(s) to the system, $\mathbf{f}$, (*e.g.* earthquake, wind, or waves) can be an additional source of uncertainty.

A mathematical treatment of the analysis of nonlinear systems can be found in [3, 4] while an engineering view of the relevant analyses procedures can be found in [5–8]. A general review of the available analysis techniques and applications of stochastic structural dynamics can also be found in the literature [9–20]. In addition, a combined treatment of nonlinear dynamics along with statistics and stochastic mechanics can be found in [21, 22].

The research presented herein is targeted at systems that possess a large number of degrees-of-freedom, but the nonlinear and uncertain substructures are spatially localized. Depending upon the presence of nonlinearity and/or uncertainty, the output of the system can be either nonlinear, uncertain, or both. Table 1.1 presents an exhaustive list of scenarios that can be encountered while modeling real engineering systems. It should be noted that $\mathcal{S}$ without any arguments is assumed to represent a

Table 1.1: Types of systems

System		Deterministic Input	Uncertain Input
Deterministic	Linear	$\mathcal{S} = \mathcal{S}$ $\mathbf{f}$ deterministic $\mathbf{y}$ deterministic	$\mathcal{S} = \mathcal{S}$ $\mathbf{f}$ uncertain $\mathbf{y}$ uncertain
	Nonlinear	$\mathcal{S} = \mathcal{S}(\mathbf{p}_n)$ $\mathbf{f}$ deterministic $\mathbf{y}$ deterministic	$\mathcal{S} = \mathcal{S}(\mathbf{p}_n)$ $\mathbf{f}$ uncertain $\mathbf{y}$ uncertain
Uncertain	Linear	$\mathcal{S} = \mathcal{S}(\mathbf{p}_u)$ $\mathbf{f}$ deterministic $\mathbf{y}$ uncertain	$\mathcal{S} = \mathcal{S}(\mathbf{p}_u)$ $\mathbf{f}$ uncertain $\mathbf{y}$ uncertain
	Nonlinear	$\mathcal{S} = \mathcal{S}(\mathbf{p}_u, \mathbf{p}_n)$ $\mathbf{f}$ deterministic $\mathbf{y}$ uncertain	$\mathcal{S} = \mathcal{S}(\mathbf{p}_u, \mathbf{p}_n)$ $\mathbf{f}$ uncertain $\mathbf{y}$ uncertain

linear and deterministic system throughout the text.

Local nonlinearities can result from sources, such as nonlinear dampers (electro-rheological, magneto-rheological, hysteretic dampers, *etc.*) in some substructures to restrain the deflections of some of the degrees-of-freedom; nonlinear stiffnesses, which can arise due to the change in the material properties due to localized damage (*e.g.* local buckling); nonlinear bearing supports; or plastic hinges forming under

extreme loading conditions. Local uncertainties can also arise via similar sources; if, for instance, the changes in system parameters due to damage in a small portion of the structure are not known deterministically. It is well established that while the system maybe mostly linear, the introduction of these local features render the response and hence the output of the entire system nonlinear and/or nondeterministic.

Section 1.1 reviews some commonly employed uncertainty quantification methods. Section 1.2 reviews the mathematical techniques utilized to reduce the size of the mathematical model of a given system and focuses on efficient computational methods for locally modified systems. Section 1.3 discusses the state-of-the-art concerning the application of parallel computing to uncertainty quantification procedures. Finally, Section 1.4 outlines the organization of the remainder of the dissertation.

1.1 Uncertainty Quantification Methods

Uncertainty quantification methods, in general, can be classified as sampling-based and non sampling-based methods. The first and most basic sampling based method is the classical Monte Carlo method. The term 'Monte Carlo' was coined in the 1940s at the Los Alamos National Laboratory [23]. Monte Carlo methods function by generating independent realizations of random inputs based on their prescribed probability function (or experimental observations) and then solving a deterministic problem for each realization of the input. Relevant statistics, such as mean and variance, can then be computed using sampling averages from the generated ensemble of outputs. The primary advantage of sampling-based methods is their applicability in limited-data problems and their ability to utilize existing deterministic system solvers to quantify the uncertainty. However, the drawback with the classical Monte Carlo method is its slow convergence rate and the dependence of this convergence rate on the behavior of higher-order moments of the associated random fields.

The classical Monte Carlo method possesses a convergence rate of $N_s^{-1/2}$ for the estimation of the mean field, where N_s is the number of samples utilized, if the second-order moments of the associated random field are well-behaved and smooth; otherwise, the convergence rate is reduced [24, 25]. Attempts have been made to accelerate the convergence rate of the Monte Carlo method by improving the sampling

technique utilized to populate the input random processes. Neyman [26] introduced the notion of stratified sampling as opposed to usual random sampling. Other improvements include Latin hypercube sampling [27–29], Bootstrap sampling [30], quasi-Monte Carlo sampling [31–33], importance sampling [34], Markov chain Monte Carlo sampling [35] and multi-scale and multi-grid Monte Carlo sampling [36–38]. A comparison of the classical Monte Carlo method with other sampling methods can be found in the literature [27, 39, 40].

In addition to their versatility in being capable of handling situations where all other methods fail to succeed, sampling-based methods are inherently amenable to parallel computing [41, 42]. Therefore, due to the widespread availability of high-performance computing resources in recent years, sampling-based methods have attracted much attention in their application to engineering mechanics and dynamics despite their slow convergence rate. Moreover, the usability of structural reanalysis methods when performing uncertainty quantification via Monte Carlo methods also warrants the application of sampling-based methods in situations where the uncertainty is spatially localized in the system. Hence, the computational methods developed in this work are aimed primarily towards the use of sampling-based methods for uncertainty quantification. Owing to their relationship with reanalysis methods, mentioned above; the developed methods can be readily used in other application areas such as structural design, design optimization and sensitivity analysis. A brief review of non sampling-based uncertainty quantification methods will be provided in the remainder of this section.

Non sampling-based methods include moment-equation based methods, perturbation methods, operator-based methods, and generalized polynomial chaos based methods. Moment-equation based methods are perhaps the most straightforward to apply, as the statistical quantities of interest can be formulated directly from the equations governing the behavior of the deterministic system. However, they typically suffer from the 'closure problem', *i.e.*, the requirement of the knowledge of higher statistical moments in order to compute the lower ones. When the system is deterministic and only the input to the system is uncertain, such methods have been studied extensively and their treatment can be found in standard texts on random vibration [43, 44].

Perturbation methods have also been previously studied. Their application to solving problems involving random fields is an extension of their application in nonlinear, deterministic analysis [45]. Random fields are expanded via a Taylor series around their mean and truncated at a certain order. This approach has been applied extensively in engineering fields and led to the development of the probabilistic finite element method [46–49]. These methods, however, suffer from the traditional drawback of perturbation type methods; the magnitude of the uncertainty has to be sufficiently small for them to perform well. As the magnitude of the uncertainty increases, the order of the terms after which the Taylor series can be truncated also grows thus rendering the involved mathematical operations computationally intractable in a general scenario.

Operator-based methods are based on the manipulation of the stochastic operators in the governing equations and function by computing the inverse of a given operator, if it exists, by using a Neumann series expansion [50, 51] or the weighted integral method [52, 53]. Similar to perturbation methods, operator-based methods are also restricted to small magnitude uncertainties. Moreover, their applicability is often limited to static problems.

Generalized polynomial chaos methods involve the discretization of the associated random fields using the Karhunen-Loève [54, 55] expansion employing orthogonal polynomials of the input random parameters; different families of orthogonal polynomials can be chosen based on the estimated distribution of the associated random phenomenon in order to achieve better convergence. The work of Ghanem and Spanos [56] is the seminal work in the application of these methods, especially in the analysis of mechanical systems, after the concept was first introduced by Wiener [57]. The concept of classical polynomial chaos has been generalized and is also studied under the name of spectral methods for uncertainty quantification [58, 59] and has been applied both in structural analysis procedures [60, 61] and non-structural analysis procedures [62–64]. Although generalized polynomial chaos based methods exhibit better convergence rates compared to sampling-based methods, their applicability to epistemic uncertainties is challenging.

1.2 Efficient Analysis of Large Computational Models

The determination of the response of a nonlinear model with many degrees-of-freedom is generally extremely computationally demanding. The analysis becomes even more challenging in the presence of uncertainties. However, many methods have been developed to deal with large systems which are mostly linear but have certain local features. The methods developed under this category share a deep relationship with methods developed for the reanalysis of structures. Reanalysis problems are common in areas such as structural design and optimization where a select few parameters of the systems are varied in order to obtain a safe and economical design.

As discussed earlier, two approaches can be employed to efficiently analyze large systems: (i) use of mathematical techniques to reduce the size of the computational model of the system in question, and (ii) use of parallel computing resources to render the associated computations more efficient. This section presents a brief overview of the former approach. In general, the mathematical procedures utilized to reduce the size of the computational model of a large system are termed model reduction methods.

The central idea of model reduction is to modify the mathematical model of a given system so that the resulting reduced order model is considerably 'smaller' in terms of computational expense than the original one. At the same time, this reduced order model should capture at least approximately, if not exactly, the necessary characteristics and crucial properties of the full system. A detailed review of general model reduction techniques can be found in the literature [65–68]. Model reduction methods can be categorized into several groups: (i) modal truncation methods [69, 70], (ii) Krylov subspace based methods [71–75], (iii) singular value decomposition based methods [76, 77], (iv) proper orthogonal decomposition methods [78], (v) balanced truncation method [79], and (vi) Fourier reduction method [80]. A historical perspective of the development of various model reduction methods can be gained by examining the comprehensive reference lists compiled by Genesio and Milanese [81] and Hickin and Sinha [82].

Most of these methods work well in the case of linear, deterministic systems; however, the model reduction of nonlinear uncertain systems is quite challenging. Reviews of nonlinear model reduction methods can also be found in the literature [83–87]. Considering the scope of the dissertation, efficient analysis techniques and model

reduction methods related to systems with local nonlinearities will be now discussed in greater depth. In addition, a detailed review of reanalysis techniques will also be given due to their applicability in the uncertainty quantification of systems with local uncertainties, as discussed earlier. Detailed reviews of various static and dynamic reanalysis techniques can be found in the literature [88–91]. The review presented herein is organized based on the three types of analyses pertinent to dynamical systems and addressed later in the dissertation, namely: modal analysis, frequency domain analysis and time domain analysis.

1.2.1 Modal Analysis

A variety of approaches have been proposed in the literature for the reanalysis of generalized, symmetric eigenproblems. In the survey paper by Chen [92], several of these methods are compared in terms of their accuracy, computational effort, and ease of implementation. These include the matrix perturbation method of Chen [93], the Bickford improved perturbation method [94], the perturbation/Rayleigh quotient method [95] and the Padé approximation method [96]. Other methods found in the literature to tackle the eigenproblem reanalysis problem include the iterated Shanks transformation [97], the epsilon-algorithm [98], the method of combined approximations [91, 99–102], the Rayleigh-Ritz reanalysis method [103, 104], the parametric reduced-order model (PROM) method [105], the improved PROM methods [106, 107], and the probabilistic reanalysis approach (PRRA) [108]. A new method developed by Wojtkiewicz and Gaurav [109] utilizes an enriched basis that consists of the subspectrum of a nominal structural system augmented with additional basis vectors generated from a knowledge of the structure of the system's uncertainty. These enrichment basis vectors are related to the so-called load-dependent Ritz vectors (LDRVs) as coined by Wilson *et al.* [110]. This method is discussed in more detail in Chapter 3.

1.2.2 Frequency Domain and Spectral Analysis

Efficient methods to compute the frequency domain and spectral response of a system have also been studied extensively in the past. Nasr [111] developed an efficient

frequency response computation algorithm based on the Arnoldi's algorithm [112]. Kim and Bennighof [113–115] developed a fast frequency response analysis (FFRA) procedure for partially damped large-scale structures with non-proportional viscous damping. In addition to these methods, substructure-based model order reduction methods have also been developed to improve the efficiency of the computation of the frequency response of linear dynamical systems [116–118]. While the developed methods work well when dealing with a single large system whose frequency response is to be determined, they are not readily applicable in scenarios where a family of related dynamical systems are to be analyzed. Such scenarios arise regularly in uncertainty quantification procedures. Wojtkiewicz *et al.* [119] and Gaurav and Wojtkiewicz [120] developed methods that are exact in nature and can tackle this scenario by utilizing the frequency response of a given nominal system to compute the response of related systems in a computationally efficient manner. These methods are based on the well-known Sherman-Morison [121] or the Sherman-Morison-Woodbury [122] formula and are discussed in detail in Chapter 4.

1.2.3 Time Domain Analysis

Large systems with local nonlinearities have been studied extensively in the context of mechanical systems. Most of the methods developed to obtain the response of such systems operate by converting the large linear portion of the model to a reduced-order model. Guyan and static reduction techniques were used by Rouch and Kao [123] and McLean and Hahn [124], respectively. Huang and Shiau [125] employed a polynomial expansion method for the analysis of nonlinear rotor bearing systems. Fey *et al.* [126] tackled locally nonlinear systems by first reducing the modes of a reference linear system using component mode synthesis and then using finite differences to solve the resulting two-point boundary value problems to compute periodic solutions. Zheng and Hasebe [127] developed a method to deal with large systems with localized nonlinearities; the linear portion of the system was first reduced using modal truncation and the resulting coupled, reduced nonlinear system was then solved using a modified Newmark integration scheme. Qu [128] developed reduction methods based on the dynamic condensation technique. Segalman [129] developed an approach to reduce the order of dynamical systems with localized nonlinearities based on the eigenmodes of a reference linear system and a set of basis functions to accommodate the nonlinearities.

While some of the aforementioned methods demonstrate reasonable computational efficiencies, since they are essentially model-reduction techniques based on either eigenmode or other form of decomposition, it becomes necessary to compute these modes every time such a model-reduction is to be performed. Moreover, due to the constitution of these methods, it is not possible to directly compute the response of a certain selected degrees-of-freedom without solving the entire system for all the degrees-of-freedom.

Another approach utilizes a convolution integral. The response of the locally nonlinear system can be represented as the response of a related, nominal linear system with the addition of a pseudo-force imparted on the nominal linear system due to the local nonlinearity. Several authors[130–136] have utilized this class of methods. In these methods, the displacement of the entire nonlinear system is solved for directly. For instance, Gordis and Radwick [134] studied the behavior of a plate resting on four point isolators with nonlinear stiffness behavior. The solution of the nonlinear Volterra integral equation (NVIE) governing the local structural displacement using a quadrature method demonstrated a gain in computational efficiency over a direct FEM solution; however, the results reported therein showed a synthesized response that was noticeably different from the direct finite element solution. In these convolution-based methods, the displacement of the entire nonlinear system is solved for directly. Further, it should be noted that only deterministic systems were investigated in these efforts.

Recently, Odes [137] and Wojtkiewicz [138] formulated a convolution approach for the deterministic response and sensitivity of locally nonlinear/uncertain systems employing a nonlinear Volterra equation in non-standard form which reduced the size of the governing equations to that of the nonlinear portion of the model. A standard trapezoidal quadrature rule was employed therein. Gains in computational efficiency of $3-10$ times were reported for response records of short length. For longer time records, these gains in efficiency were lost to the quadratic computational cost dependence of the trapezoidal rule with record length. Further, reduced accuracy was displayed in the case of systems with non-smooth nonlinearities.

While the potential of a non-standard form nonlinear Volterra integral equation formulation was demonstrated in [137, 138], the computational efficiency and accuracy of the solution procedure adopted for solving the associated system of NVIEs therein leaves significant space for improvement. Several types of methods have been developed

in the past for solving Volterra equations including quadrature based methods, explicit and implicit Rünge-Kutta methods and collocation-based methods [139–141].

Hairer *et al.* [140] developed a fast explicit Rünge-Kutta based method to solve convolution type NVIEs. Their main idea was to divide the convolution space into sub-spaces so that a fast Fourier transform (FFT)-based convolution can be used in part of the space, which reduced the computational cost. Recently, Isaacson and Kirby [142] developed fast collocation-based methods to solve linear VIEs. However, both these studies assumed a scalar Volterra kernel. Capobianco *et al.* [143] developed fast Rünge-Kutta methods for general NVIEs based on a Laplace domain inversion for the computation of the Volterra kernel. The main limitation of methods based on Laplace domain inversion of the Volterra kernel is the need of appropriately choosing the contours (*e.g.* Talbot's contour [144]) for the associated contour integral. One of the drawbacks of using Rünge-Kutta methods, collocation methods, or other multi-stage methods, is the necessity of the computation of the Volterra kernel at intermediate time-points (stages), which further increases the computational time. Moreover, all these methods require the knowledge of the Volterra kernel as a closed-form function of time and/or the states of the system. Oftentimes, the kernel and/or the applied load is known only as a discrete time-series rather than as a closed functional form (*e.g.* in cases of structures subjected to random excitations). A new approach developed by Gaurav *et al.* [145] develops a Newton-Gregory quadrature-based method to solve NVIEs with the following characteristics: (i) ability to compute the response of a selected subset of the full response, (ii) ability to handle both smooth and non-smooth nonlinearities, and (iii) good computational efficiency so that a large number of samples can be generated for sampling-based uncertainty quantification studies. This approach is discussed in more detail in Chapter 5.

1.3 Parallel Computing in Uncertainty Quantification

The idea of parallel computing, *i.e.*, connecting multiple processors to collectively solve a problem originated in the 1960s, the decade following the introduction of the first digital computer in 1954. Although the processors of that era were not as advanced as modern day ones, key concepts such as expected gains in computational efficiency when

parallelizing an algorithm were of interest, and metrics to measure these were developed that are still in use today, such as the Amdahl's law [146], later improved by Gustafson [147]. Flynn [148] proposed a classification of computer architecture which has been widely utilized to classify parallel computing machines and applications. He proposed four categories:

- Single Instruction, Single Data (SISD): A sequential computer which offers no parallelism, *e.g.* a uniprocessor machine like earlier PC (before multi-core processors were conceived). In terms of application, it means only one type of computational instructions is to be performed on a single set of data.

- Single Instruction, Multiple Data (SIMD): A computer that exploits multiple data streams against a single instruction stream, *e.g.* an array processor, or a graphics processing unit (GPU). In terms of application, it means one type of computational instruction being applied to multiple data sets.

- Multiple Instruction, Single Data (MISD): A computer that applies multiple sets of instructions on a single data set. This is an unusual architecture that is generally used for fault tolerance.

- Multiple Instruction, Multiple Data (MIMD): A computer that has multiple autonomous processors performing different computational instructions on different sets of data. Most of the present CPU-based parallel computers fall into this category.

Three paths may be followed to use parallel computing for uncertainty quantification procedures:

I. Parallelization of the uncertainty quantification method itself.

II. Parallelization of the methods utilized to compute the response of the associated engineering system.

III. Mixed use of the above two.

The first path is relatively straightforward, especially with regard to sampling-based methods, known to be embarrassingly parallel, and is the primary focus of the methods

developed herein. Parallel strategies for sampling-based methods have been studied previously [41, 42]. As such, the generation of different samples for the required ensemble is distributed to different computing nodes. This is an example of a SIMD type parallelization strategy, where a single instruction (response computation of the engineering system) is performed on different data sets (different models of the engineering system arising from the different samples of associated random parameters and/or inputs).

The second path usually leverages basic parallel computing primitives, such as matrix multiplication, in order to parallelize part of the algorithm utilized to compute the response of the system. Colella *et al.* [149] identified seven dwarfs, or building blocks, of parallel computing believed to be important for applications in science and engineering for at least the next decade. Asanovic *et al.* [150] later added six more to this list. These thirteen dwarfs are shown in Figure 1.3. The application of these

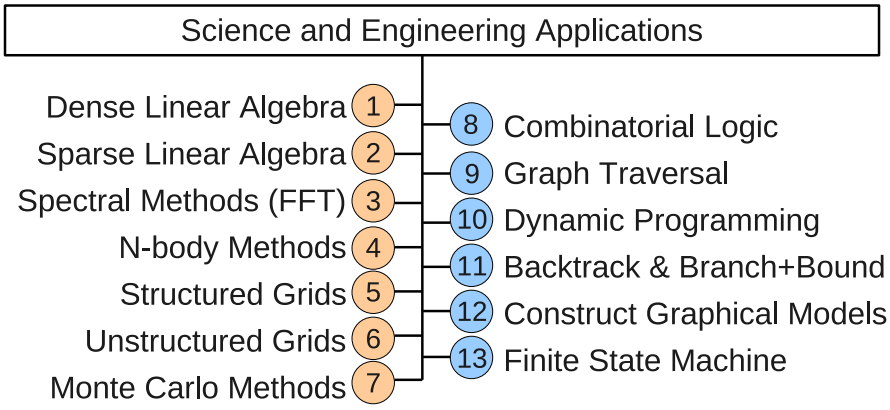


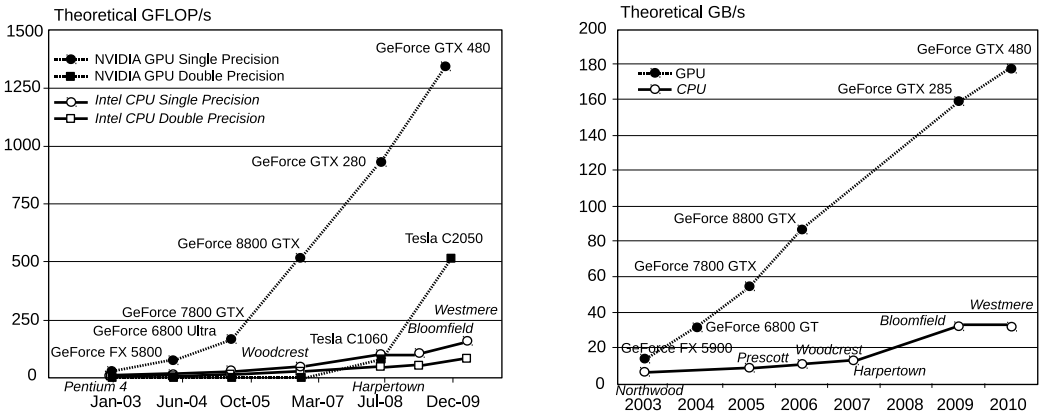
Figure 1.3: Dwarfs of parallel computing

thirteen dwarfs to compute the response of an engineering system is most effective when the size of the system's model is large. Software packages, such as DAKOTA [151, 152], provide object-oriented implementations of uncertainty quantification procedures that can be readily utilized to analyze large systems with uncertainties. DAKOTA can also accommodate parallel processing resources for large systems by exploiting shared-memory and distributed-memory CPU-based parallel implementations.

While CPU-based parallel computing has previously provided significant avenues for reducing the computational times, it provides linear speed-up at best, *i.e.*, if N_{proc} processors are used for a given algorithm, the corresponding computational time can be reduced by a factor of N_{proc}. In recent years, the use of graphics processing units (GPUs) for general purpose computing has led to a paradigm shift in parallel computing. Primarily driven by the gaming industry, GPUs have evolved from being partially programmable in 1999 to a much more economical and highly competitive alternative architecture to CPU-based parallel computing in 2011. As the degree of software control (programmability) of the GPUs has increased, its use in the solution of general purpose computing problems has spawned the discipline of general purpose GPU (GPGPU) computing. The interest of researchers in GPGPU is primarily because of four reasons: (i) better performance, (ii) anticipated evolution, (iii) cost efficiency, and (iv) energy efficiency of the GPUs.

The performance of a given algorithm on a given processor is primarily governed by two factors: peak computational capability of the processor (usually measured in Giga floating point operations per second, GFLOP/s) and memory bandwidth between the processor and its dynamic random access memory (usually measured in Gigabytes per second, GB/s). Figure 1.4 shows a comparison of these two metrics of computational performance for GPUs versus contemporary CPUs. While the peak performance of current CPUs is less than 250 GFLOP/s and 40 GB/s, GPUs have not only broken the teraflop barrier but also outperform the CPUs in memory bandwidth by a factor of almost five. Apart from depicting a clear computational edge of the GPUs compared to the CPUs, Figure 1.4 highlights another differentiating feature of the GPUs, their chronological development. While GPUs show an almost linear trend with time in their development, CPUs await a breakthrough in hardware development to take them out of the saturation point that started to be displayed in 2003.

The evolution gap between the CPUs and the GPUs can be attributed to their differing design philosophies. When the CPUs stopped developing in accordance with the Moore's law [154] in 2003, multi-core CPUs were conceived. While multi-core chips provided exploitable parallelism to applications, their primary objective is to maintain, and possibly increase the performance of sequential programs. GPUs, on the other hand, were initially designed to cater to computer gaming and graphics rendering which require



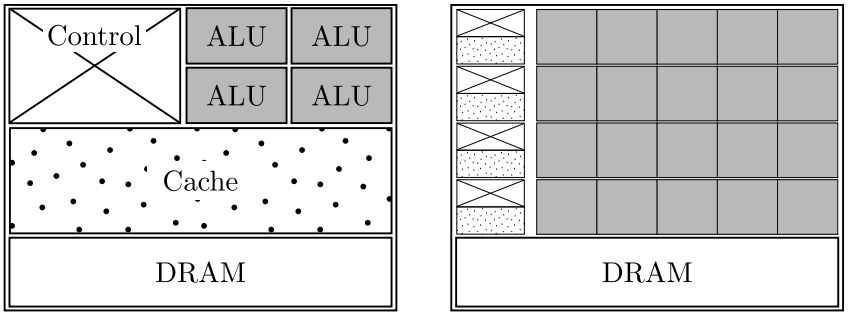
(a) FLOP/s (b) Bandwidth

Figure 1.4: Evolution of GPUs compared to CPUs.
(adapted from [153])

a large number of relatively lightweight computations. Thus, GPUs evolved based on the so-called many-core philosophy, which employs a large number of small computational cores on the same chip. These different design philosophies are shown in Figure 1.5. The size of the cores allows GPUs to economically contain a larger number of cores compared to multi-core CPUs. Table 1.2 summarizes the principal differences between these two design philosophies. The economic viability along with the size of the cores on the

Table 1.2: Multi-core vs Many-core

Multi-core (CPU)	**Many-core (GPU)**
Principal objective: maintain execution speed of sequential programs	*Principal objective*: focus on execution throughput of parallel applications
Contains small number of large cores	Contains large number of small cores
Sophisticated control logic to control threads	Simple control logic to control threads
More expensive to initiate and manage threads	Less expensive to initiate and manage threads



(a) CPU Architecture (b) GPU Architecture

Figure 1.5: Multi-core vs Many-core Design Philosophy (adapted from [153])

Table 1.3: Desktop-level Comparison between CPU and GPU

NVIDIA Tesla C1060	**Intel Xeon E5530**
Number of computational cores	
240	4 (8 with hyper-threading)
Peak theoretical performance	
933 GFLOP/s (SP) 125 GFLOP/s (DP) 100 GB/s	120 GFLOP/s (SP) 80 GFLOP/s (DP) 25 GB/s
Cost efficiency	
$1.3 per GFLOP/s	$4.5 per GFLOP/s
Energy efficiency	
0.4 watts per GFLOP/s	0.7 watts per GFLOP/s

GPUs have a direct impact on their cost and energy efficiency. A comparison between Intel's Xeon E5530 quad-core processor [155] and NVIDIA's Tesla C1060 GPU [156] in Table 1.3 shows that the former yields a cost efficiency of about \$4.5 per GFLOP/s and an energy efficiency of about 0.7 watts per GFLOP/s while the latter yields about \$1.3 per GFLOP/s and 0.2 watts per GFLOP/s. These specific models of the CPU and the GPU were chosen for efficiency comparisons because this hardware has been utilized for performing the numerical studies presented later in the dissertation. These models form a representative pair for comparing CPU and GPU performances because of their equivalence in market price and convenient availability in desktop computers.

GPUs are best suited for one of the very commonly employed parallelization strategies, SIMD, because all the cores in each of their multiprocessors are single instruction multiple thread (SIMT) cores, *i.e.*, all the cores execute the same set of instructions, but with different data. A large number of processor cores (*e.g.* 240 in the NVIDIA Tesla) along with a large number of active threads is the key to a GPU's performance. Threads execute in groups, called warps, and the execution alternates between active warps with warps becoming temporarily inactive when waiting for data. This feature hides the memory latency to a great extent. Memory latency and the availability of only a small amount of shared memory among the threads are the two prime challenges for GPU algorithm developers. From a software point of view, programming on GPUs has become considerably easier since the introduction of the NVIDIA CUDA™ programming environment, which is much like the programming language C.

Initially, GPGPU was aimed towards the thirteen dwarfs as discussed earlier. Dense linear algebra and FFT algorithms have already matured into optimized routines and are available as vendor libraries for GPUs, CUBLAS [157] and CUFFT [158], respectively. CUBLAS provides GPU implementations of all the basic linear algebra subroutines (BLAS) while CUFFT provides GPU implementations of one-, two-, and three-dimensional fast Fourier transforms. While these implementations are meant for pure GPU applications, hybrid algorithms have also been developed. Tomov *et al.* [159] have developed a suite of hybrid implementations of the equivalents of BLAS and LAPACK routines, which are available as a library, called MAGMA [160].

Bolz *et al.* [161] developed conjugate gradient and multi-grid sparse matrix solvers for GPUs. Later, Bell and Garland [162] and Vazquez *et al.* [163] developed efficient sparse matrix-vector product implementations. Tomov *et al.* [164] developed GPU implementation of probability-based simulations (Ising and percolation models). Recently, Tian and Benkrid [165] developed a GPU implementation of the Mersenne-Twister random number generation algorithm. The use of GPUs in science and engineering computational simulations can also be found in the literature. So [166] developed time domain computational electromagnetics algorithms for GPU-based computers. Zhao [167] developed GPU-accelerated PDE solver using the lattice Boltzmann model. Walsh *et al.* [168] developed GPU implementations of algorithms pertinent to geoscience and engineering system simulations. Januszewski and Kostur [169] developed a GPU solver for stochastic differential equations.

Detailed surveys and lists of additional applications of GPGPU in computational science and engineering can be found in the literature [170–176]. Gaurav and Wojtkiewicz [177] have developed GPU implementations of several algorithms pertinent to uncertainty analysis of dynamical and mechanical systems. These implementations are discussed in detail in Chapter 6.

1.4 Outline of the Dissertation

Chapter 2 summarizes the theoretical background related to the analysis of linear dynamical systems required to facilitate the developments in the remainder of the dissertation. Analytical and numerical methods employed to analyze systems with nonlinearities and uncertain inputs and/or parameters are also discussed.

Chapter 3 discusses the development of a novel, efficient computational methodology for the modal analysis of undamped linear systems with local stiffness uncertainties. Numerical examples are presented to demonstrate the error behavior, computational efficiency and applicability of the proposed method.

Chapter 4 develops an exact method for the frequency domain analysis of a linear, time-invariant dynamical system with local uncertainties in damping and stiffness characteristics. The proposed method can be utilized for either the computation of the transfer function matrix or the spectral response of a family of related systems in

a computationally efficient manner. Numerical examples are presented to demonstrate the computational efficiency and applicability of the proposed method.

Chapter 5 presents the development of a computationally efficient method to analyze time-invariant dynamical systems with local nonlinearities and uncertainties. Numerical examples are presented to demonstrate the error behavior, computational efficiency and applicability of the proposed method.

Chapter 6 explores the use of a graphics processing unit (GPU) to perform uncertainty analysis of computational models in an efficient manner. Strategies to parallelize existing procedures along with approaches to implement parallel algorithms on GPUs are discussed. GPU implementations of five techniques, frequently used in uncertainty quantification in computational mechanics, are developed that show the usability of GPUs in this area. In addition, a GPU implementation of the efficient time domain method presented in Chapter 5 is also presented.

Chapter 7 concludes the dissertation with a mention of broader impacts of the presented work and future work related to the methods developed in the dissertation. Appendix A lists the mathematical symbols and acronyms used throughout the dissertation.

Chapter 2

Theoretical Background

There are several methods to derive the equations of motion of a dynamical system using the law of conservation of energy in one of its various available forms (*e.g.* Hamilton's principle and the principle of virtual work). The equations of motion obtained are usually in the form of partial differential equations (PDEs) in spatial and time coordinates. Spatial discretization techniques, such as the finite element method, are generally applied to convert these PDEs into ordinary differential equations (ODEs) in time. Methods, such as direct integration, numerical quadrature, *etc.* can then be utilized to solve these ODEs.

This chapter briefly presents the basic theory of the analysis of dynamical systems starting with linear, time-invariant (LTI), deterministic systems followed by systems with inherent uncertainties and nonlinearities. Closed form analytical solutions for the response are provided wherever available. Pertinent numerical methods along with their computational complexity, numerical stability and convergence are also discussed.

2.1 Linear, Time-Invariant (LTI) Systems Theory

An LTI dynamical system with n degrees-of-freedom (DOFs) in discretized space can be represented as a system of n-coupled ODEs as

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{D}\dot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{B}\mathbf{f}(t), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \dot{\mathbf{x}}(0) = \dot{\mathbf{x}}_0 \tag{2.1}$$

where $\mathbf{M}, \mathbf{D}, \mathbf{K} \in \mathbb{R}^{n \times n}$ are the mass, the damping and the stiffness matrices of the system, respectively, $\mathbf{x} \in \mathbb{R}^n$ is the displacement vector, $\mathbf{B} \in \mathbb{R}^{n \times N_f}$ is an influence matrix, $\mathbf{f} \in \mathbb{R}^{N_f}$ is the vector of external excitations (inputs), and $(\dot{\ })$ denotes differentiation with respect to time. Usually, the mass matrix is positive definite and the stiffness matrix is positive semi-definite and the same has been assumed throughout this dissertation. Within the context of civil structures, the stiffness matrix can be assumed to be positive definite because structures are usually constrained in a way that prevents rigid-body motion. In addition, it is also assumed that the mass, the stiffness and the damping matrices are symmetric. The system of (2.1) can alternatively be represented in state-space as

$$\dot{\underline{\mathbf{x}}}(t) = \mathbf{A}\,\underline{\mathbf{x}}(t) + \underline{\mathbf{B}}\,\mathbf{f}(t), \quad \underline{\mathbf{x}}(t_0) = \underline{\mathbf{x}}_0 \tag{2.2}$$

where $\mathbf{A} \in \mathbb{R}^{\underline{n} \times \underline{n}}$ is the system matrix, $\underline{\mathbf{B}} \in \mathbb{R}^{\underline{n} \times N_f}$ is an influence matrix, $\underline{\mathbf{x}} \in \mathbb{R}^{\underline{n}}$ is the state-vector, and $\mathbf{f} \in \mathbb{R}^{N_f}$ is again the input vector, and where

$$\underline{\mathbf{x}} = \begin{Bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{Bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \mathbf{0}_{n \times n} & \mathbf{I}_{n \times n} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{D} \end{bmatrix}, \quad \underline{\mathbf{B}} = \begin{bmatrix} \mathbf{0}_{n \times N_f} \\ \mathbf{M}^{-1}\mathbf{B} \end{bmatrix}, \quad \underline{n} = 2n. \tag{2.3}$$

The desired output of the system is oftentimes written as a linear combination of its states as

$$\mathbf{y}(t) = \mathbf{C}_0\mathbf{x}(t) + \mathbf{C}_1\dot{\mathbf{x}}(t) + \mathbf{C}_2\ddot{\mathbf{x}}(t) \tag{2.4}$$

$$= \underline{\mathbf{C}}_0\underline{\mathbf{x}}(t) + \underline{\mathbf{C}}_1\dot{\underline{\mathbf{x}}}(t) \tag{2.5}$$

where $\mathbf{y} \in \mathbb{R}^{N_o}$ is the desired output. $\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2 \in \mathbb{R}^{N_o \times n}$ and $\underline{\mathbf{C}}_0, \underline{\mathbf{C}}_1 \in \mathbb{R}^{N_o \times \underline{n}}$ are output matrices that combine the required states to generate the desired output while N_o denotes the desired number of outputs.

2.1.1 Modal Analysis

Modal analysis of undamped dynamical systems is crucial in the response analysis of LTI systems. In certain scenarios, the modal solution of a given system can be utilized to obtain corresponding closed-form analytical solutions. The free vibration of an undamped LTI system can be written from (2.1) as

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{0}_{n \times 1} \tag{2.6}$$

after eliminating the damping and the forcing terms, respectively. The eigenvalues and the eigenvectors of the system can be determined by solving the generalized eigenvalue problem given by

$$\mathbf{K}\boldsymbol{\Phi} = \mathbf{M}\boldsymbol{\Phi}\boldsymbol{\Lambda} \tag{2.7}$$

where $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times n}$ is a diagonal matrix consisting of the eigenvalues of the system, and $\boldsymbol{\Phi} \in \mathbb{R}^{n\times n}$ is the modal matrix consisting of mass-normalized eigenvectors of the system.

$$\boldsymbol{\Lambda} = \mathrm{diag}\big[\lambda_1, \lambda_2, \ldots, \lambda_n\big]\,, \quad \boldsymbol{\Phi} = \begin{bmatrix} \phi_{1,1} & \phi_{2,1} & \cdots & \phi_{n,1} \\ \phi_{1,2} & \phi_{2,2} & \cdots & \phi_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{1,n} & \phi_{2,n} & \cdots & \phi_{n,n} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 & \ldots \boldsymbol{\phi}_n \end{bmatrix}. \tag{2.8}$$

If both the mass and the stiffness matrices are positive definite, the eigenvalues computed from (2.7) will all be real and positive. Moreover, the eigenvectors can be shown to be linearly independent and orthogonal with respect to the mass matrix. As the eigenvectors of the system are assumed to be mass-normalized, the following relation also holds:

$$\boldsymbol{\Phi}^{\mathsf{T}}\mathbf{M}\boldsymbol{\Phi} = \mathbf{I}_{n\times n}. \tag{2.9}$$

Pre-multiplying (2.7) by $\boldsymbol{\Phi}^{\mathsf{T}}$ and using (2.9) yields

$$\boldsymbol{\Phi}^{\mathsf{T}}\mathbf{K}\boldsymbol{\Phi} = \boldsymbol{\Lambda}. \tag{2.10}$$

Since the eigenvectors are linearly independent, they form a basis in the space $\mathbb{R}^n$ which enables the representation of the solution of the system in (2.1) as

$$\mathbf{x}(t) = \sum_{i=1}^{n} \boldsymbol{\phi}_i q_i(t) = \boldsymbol{\Phi}\mathbf{q}(t) \tag{2.11}$$

where $q_i(t)$ are the modal coordinates and (2.11) defines a linear transformation between the physical coordinates, $\mathbf{x}$, and the modal coordinates, $\mathbf{q}$. Using this transformation, (2.1) can be transformed into an equivalent LTI system governing the evolution of the modal coordinates given by

$$\ddot{\mathbf{q}}(t) + \boldsymbol{\Phi}^{\mathsf{T}}\mathbf{D}\boldsymbol{\Phi}\dot{\mathbf{q}}(t) + \boldsymbol{\Lambda}\mathbf{q}(t) = \boldsymbol{\Phi}^{\mathsf{T}}\mathbf{B}\mathbf{f}(t). \tag{2.12}$$

An analogous eigenvalue problem can be written for the state-space representation given by

$$\mathbf{A}\underline{\boldsymbol{\Phi}} = \underline{\boldsymbol{\Phi}}\,\underline{\boldsymbol{\Lambda}} \tag{2.13}$$

where $\underline{\mathbf{\Lambda}} \in \mathbb{R}^{\underline{n}\times\underline{n}}$ is a diagonal matrix consisting of the eigenvalues of the system and $\underline{\mathbf{\Phi}} \in \mathbb{R}^{\underline{n}\times\underline{n}}$ is the modal matrix comprising of the eigenvectors of the system.

$$\underline{\mathbf{\Lambda}} = \mathrm{diag}\big[\underline{\lambda}_1, \underline{\lambda}_2, \ldots, \underline{\lambda}_{\underline{n}}\big]\,, \quad \underline{\mathbf{\Phi}} = \begin{bmatrix} \underline{\phi}_{1,1} & \underline{\phi}_{2,1} & \cdots & \underline{\phi}_{\underline{n},1} \\ \underline{\phi}_{1,2} & \underline{\phi}_{2,2} & \cdots & \underline{\phi}_{\underline{n},2} \\ \vdots & \vdots & \ddots & \vdots \\ \underline{\phi}_{1,n} & \underline{\phi}_{2,n} & \cdots & \underline{\phi}_{\underline{n},\underline{n}} \end{bmatrix} = \begin{bmatrix} \underline{\boldsymbol{\phi}}_1 & \underline{\boldsymbol{\phi}}_2 & \cdots \underline{\boldsymbol{\phi}}_{\underline{n}} \end{bmatrix}. \quad (2.14)$$

When the eigenvalues of $\mathbf{A}$ are all distinct, the eigenvectors are linearly independent and the inverse of the modal matrix, $\underline{\mathbf{\Phi}}$, exists. In such a case, the system matrix, $\mathbf{A}$, can be represented as the following similarity transformation:

$$\mathbf{A} = \underline{\mathbf{\Phi}}\,\underline{\mathbf{\Lambda}}\,\underline{\mathbf{\Phi}}^{-1}. \quad (2.15)$$

Again, the response of the system can be written in terms of the basis formed by $\underline{\mathbf{\Phi}}$ as

$$\underline{\mathbf{x}}(t) = \sum_{i=1}^{\underline{n}} \underline{\boldsymbol{\phi}}_i \underline{q}_i(t) = \underline{\mathbf{\Phi}}\underline{\mathbf{q}}(t) \quad (2.16)$$

and (2.2) can be transformed using (2.16) to obtain the LTI system governing the evolution of the modal coordinates of the state-space representation given by

$$\dot{\underline{\mathbf{q}}}(t) = \underline{\mathbf{\Lambda}}\underline{\mathbf{q}}(t) + \underline{\mathbf{\Phi}}^{-1}\underline{\mathbf{B}}\,\underline{\mathbf{f}}(t). \quad (2.17)$$

Moreover, when an undamped system as in (2.6) is represented in state-space using (2.3), the following relations can be derived between the eigenvalues and the eigenvectors of the configuration-space and the state-space representations of the system:

$$\underline{\lambda}_i^2 = -\lambda_i; \quad \text{and} \quad \underline{\boldsymbol{\phi}}_i = \begin{bmatrix} \boldsymbol{\phi}_i^{\mathsf{T}} & \underline{\lambda}_i \boldsymbol{\phi}_i^{\mathsf{T}} \end{bmatrix}^{\mathsf{T}}. \quad (2.18)$$

2.1.2 Time Domain Analysis

In configuration space, exact analytical solutions of (2.1) can be written when the system is subjected to classical damping, *i.e.*, the term $\mathbf{\Phi}^{\mathsf{T}}\mathbf{D}\mathbf{\Phi}$ in the transformed system of equations, (2.12), is diagonal. The damping is termed classical when either of the following two conditions hold:

- $\mathbf{D} = \alpha\mathbf{M} + \beta\mathbf{K}$ where α and β are real numbers. In such a case, $\mathbf{\Phi}^{\mathsf{T}}\mathbf{D}\mathbf{\Phi} = \alpha\mathbf{I}_{n\times n} + \beta\mathbf{\Lambda}$ is diagonal. When $\mathbf{D}$ is of this form, the system is said to possess Rayleigh damping [178].

- $\mathbf{D}\mathbf{M}^{-1}\mathbf{K} = \mathbf{K}\mathbf{M}^{-1}\mathbf{D}$ [179].

In the case of classical damping, $\mathbf{\Phi}^{\mathsf{T}}\mathbf{D}\mathbf{\Phi}$ can be written as

$$\mathbf{\Phi}^{\mathsf{T}}\mathbf{D}\mathbf{\Phi} = \mathrm{diag}\big[2\zeta_1\omega_{n_1}, 2\zeta_2\omega_{n_2}, \ldots, 2\zeta_n\omega_{n_n}\big] \tag{2.19}$$

where $\omega_{n_i}^2 = \lambda_i$ and ζ_i represents i^{th} damping ratio of the system. Thus, (2.12) can now be written as a system of n decoupled ODEs as

$$\ddot{q}_i(t) + 2\zeta_i\omega_{n_i}\dot{q}_i(t) + \omega_{n_i}^2 q_i(t) = \boldsymbol{\phi}_i^{\mathsf{T}}\mathbf{f}(t) = f_i(t). \tag{2.20}$$

Further, the general analytical solution of (2.20) can be written using the convolution integral as

$$q_i(t) = g_i(t-t_0)q_{i,0} + h_i(t-t_0)\dot{q}_{i,0} + \int_{t_0}^{t} h_i(t-\tau)f_i(\tau)\mathrm{d}\tau \tag{2.21}$$

where $q_{i,0}$ and $\dot{q}_{i,0}$ are the initial conditions imposed on the ith modal coordinate and can be obtained as the ith component of the vectors

$$\mathbf{q}_0 = \mathbf{\Phi}^{-1}\mathbf{x}_0 = \mathbf{\Phi}^{\mathsf{T}}\mathbf{M}\mathbf{x}_0 \tag{2.22}$$

$$\dot{\mathbf{q}}_0 = \mathbf{\Phi}^{-1}\dot{\mathbf{x}}_0 = \mathbf{\Phi}^{\mathsf{T}}\mathbf{M}\dot{\mathbf{x}}_0 \tag{2.23}$$

and where $g_i(t)$ and $h_i(t)$ are defined as

$$g_i(t) = \begin{cases} e^{-\zeta_i\omega_{n_i}t}\left[\cos\left(\omega_{d_i}t\right) + \frac{\omega_{n_i}\zeta_i}{\omega_{d_i}}\sin\left(\omega_{d_i}t\right)\right] & \text{under-damped} \\ (1+\omega_{n_i}t)e^{-\omega_{n_i}t} & \text{critically damped} \\ e^{-\zeta_i\omega_{n_i}t}\left[\cosh\left(\omega_{d_i}^{*}t\right) + \frac{\omega_{n_i}\zeta_i}{\omega_{d_i}^{*}}\sinh\left(\omega_{d_i}^{*}t\right)\right] & \text{over-damped} \end{cases} \tag{2.24}$$

$$h_i(t) = \begin{cases} \frac{1}{\omega_{d_i}}e^{-\zeta_i\omega_{n_i}t}\sin\left(\omega_{d_i}t\right) & \text{under-damped} \\ te^{-\omega_{n_i}t} & \text{critically damped} \\ e^{-\zeta_i\omega_{n_i}t}\sinh\left(\omega_{d_i}^{*}t\right) & \text{over-damped} \end{cases} \tag{2.25}$$

$$i = 1, 2, \ldots, n$$

where $\omega_{d_i} = \omega_{n_i}\sqrt{1-\zeta_i^2}$ and $\omega_{d_i}^* = \omega_{n_i}\sqrt{\zeta_i^2-1}$ are the damped natural frequencies of the under-damped and the over-damped system, respectively. A system is termed under-damped, critically damped, or over-damped based on the following criterion:

$$\zeta_i = \begin{cases} < 1 & \text{under-damped} \\ = 1 & \text{critically damped}\,, \\ > 1 & \text{over-damped} \end{cases} \quad i = 1, 2, \ldots, n. \tag{2.26}$$

The response of the physical system can be computed using (2.11) once all the components of the modal response have been computed as

$$\mathbf{x}(t) = \mathbf{g}(t-t_0)\mathbf{M}\mathbf{x}_0 + \mathbf{h}(t-t_0)\mathbf{M}\dot{\mathbf{x}}_0 + \int_{t_0}^{t} \mathbf{h}(t-\tau)\mathbf{f}(\tau)\mathrm{d}\tau \tag{2.27}$$

where $\mathbf{h}(t) \in \mathbb{R}^{n\times n}$ is the impulse response matrix defined as

$$\mathbf{h}(t) = \mathbf{\Phi}\mathrm{diag}\big[h_1(t), h_2(t), \ldots, h_n(t)\big]\,\mathbf{\Phi}^{\mathsf{T}} \tag{2.28}$$

and

$$\mathbf{g}(t) = \mathbf{\Phi}\mathrm{diag}\big[g_1(t), g_2(t), \ldots, g_n(t)\big]\,\mathbf{\Phi}^{\mathsf{T}}. \tag{2.29}$$

When the system is non-classically damped, closed form analytical expressions for the response of the system are usually not available. In such cases, certain methods can be applied to approximate the solution of the system including: (i) neglecting the off-diagonal terms of the matrix $\mathbf{\Phi}^{\mathsf{T}}\mathbf{D}\mathbf{\Phi}$ when they are much smaller than the diagonal elements, (ii) iterative solution process [180], and (iii) perturbation solution approach [181].

The state-space formulation (2.2) can be utilized to write analytical solutions of such systems as

$$\underline{\mathbf{x}}(t) = \underline{\mathbf{h}}(t-t_0)\underline{\mathbf{x}}_0 + \int_{t_0}^{t} \underline{\mathbf{h}}(t-\tau)\mathbf{B}\underline{\mathbf{f}}(\tau)\mathrm{d}\tau \tag{2.30}$$

where $\underline{\mathbf{h}}(t) \in \mathbb{R}^{\underline{n}\times\underline{n}}$ is the impulse response matrix of the state-space representation defined as

$$\underline{\mathbf{h}}(t) = e^{\mathbf{A}t} \tag{2.31}$$

and

$$e^{\mathbf{A}t} = \mathbf{I}_{\underline{n}\times\underline{n}} + \mathbf{A}t + \frac{t^2}{2!}\mathbf{A}^2 + \frac{t^3}{3!}\mathbf{A}^3 + \ldots. \tag{2.32}$$

When $\mathbf{A}$ has $\underline{n}$ distinct eigenvalues, it admits a representation as in (2.15) as discussed earlier. In such a case, (2.15) can be utilized to compute the matrix exponential in (2.32) as

$$e^{\mathbf{A}t} = \underline{\mathbf{\Phi}} e^{\underline{\mathbf{\Lambda}}t} \underline{\mathbf{\Phi}}^{-1} \tag{2.33}$$

where

$$e^{\underline{\mathbf{\Lambda}}t} = \mathrm{diag}\Big[e^{\underline{\lambda}_1 t}, e^{\underline{\lambda}_2 t}, \ldots, e^{\underline{\lambda}_{\underline{n}} t}\Big] \tag{2.34}$$

can be computed trivially as it involves only scalar exponentials.

2.1.3 Frequency Domain Analysis

The equations of motion, (2.1) and (2.2), can be transformed to the frequency domain by applying the Laplace transform. The Laplace transform of a function, $f(t); t \geq 0$, is defined as

$$F(s) = \mathscr{L}[f(t)] = \int_0^\infty e^{-st} f(t) \mathrm{d}t \tag{2.35}$$

where the parameter s is a complex number.

Further, by defining the following Laplace transforms:

$$\mathscr{L}[\mathbf{f}(t)] = \mathbf{F}(s) \qquad \mathscr{L}[\underline{\mathbf{f}}(t)] = \underline{\mathbf{F}}(s) \tag{2.36}$$

$$\mathscr{L}[\mathbf{x}(t)] = \mathbf{X}(s) \qquad \mathscr{L}[\underline{\mathbf{x}}(t)] = \underline{\mathbf{X}}(s) \tag{2.37}$$

$$\mathscr{L}[\dot{\mathbf{x}}(t)] = s\mathbf{X}(s) - \mathbf{x}_0 \qquad \mathscr{L}[\underline{\dot{\mathbf{x}}}(t)] = s\underline{\mathbf{X}}(s) - \underline{\mathbf{x}}_0 \tag{2.38}$$

$$\mathscr{L}[\ddot{\mathbf{x}}(t)] = s^2\mathbf{X}(s) - s\dot{\mathbf{x}}_0 - \mathbf{x}_0 \tag{2.39}$$

the frequency domain equivalent of (2.1) with $t_0 = 0$ can be written as

$$\left(\mathbf{M}s^2 + \mathbf{D}s + \mathbf{K}\right)\mathbf{X}(s) = \mathbf{F}(s) + (\mathbf{M} + \mathbf{D})\mathbf{x}_0 + s\mathbf{M}\dot{\mathbf{x}}_0. \tag{2.40}$$

The frequency domain response of the system can then be written as

$$\mathbf{X}(s) = \mathbf{H}(s)\left[\mathbf{F}(s) + (\mathbf{M} + \mathbf{D})\mathbf{x}_0 + s\mathbf{M}\dot{\mathbf{x}}_0\right] \tag{2.41}$$

where $\mathbf{H}(s) \in \mathbb{C}^{n \times n}$ is the system's transfer function matrix given by

$$\mathbf{H}(s) = \left(\mathbf{M}s^2 + \mathbf{D}s + \mathbf{K}\right)^{-1} = \mathscr{L}[\mathbf{h}(t)]\,. \tag{2.42}$$

The frequency domain equivalent of (2.2) can be written in a similar manner as

$$(s\mathbf{I} - \mathbf{A})\,\underline{\mathbf{X}}(s) = \mathbf{B}\underline{\mathbf{F}}(s) + \underline{\mathbf{x}}_0 \tag{2.43}$$

and the frequency domain response can be written as

$$\underline{\mathbf{X}}(s) = \underline{\mathbf{H}}(s)\left[\mathbf{B}\underline{\mathbf{F}}(s) + \underline{\mathbf{x}}_0\right] \tag{2.44}$$

where $\underline{\mathbf{H}}(s) \in \mathbb{C}^{\underline{n}\times\underline{n}}$ is the system's transfer function matrix given by

$$\underline{\mathbf{H}}(s) = (s\mathbf{I} - \mathbf{A})^{-1} = \mathscr{L}[\underline{\mathbf{h}}(t)]\,. \tag{2.45}$$

2.2 LTI Systems Subjected to Random Inputs

This section will present formulations to compute the response of LTI systems subjected to random inputs. The system is assumed to be deterministic. However, the initial conditions of the system are allowed to be uncertain. Moreover, the uncertainties in the initial conditions and that in the input to the system are assumed to be uncorrelated.

The mean response of the system can be computed by applying the expectation operator to (2.27) yielding

$$\boldsymbol{\mu}_{\mathbf{x}}(t) = \mathbb{E}[\mathbf{x}(t)] = \mathbf{g}(t-t_0)\mathbf{M}\mathbb{E}[\mathbf{x}_0] + \mathbf{h}(t-t_0)\mathbf{M}\mathbb{E}[\dot{\mathbf{x}}_0] + \int_{t_0}^{t} \mathbf{h}(t-\tau)\mathbb{E}[\mathbf{f}(\tau)]\,\mathrm{d}\tau \tag{2.46}$$

and similarly, the covariance response can be written as

$$\begin{aligned}\boldsymbol{\Gamma}_{\mathbf{xx}}(t_1,t_2) &= \mathbb{E}\Big[\{\mathbf{x}(t_1) - \boldsymbol{\mu}_{\mathbf{x}}(t_1)\}\,\{\mathbf{x}(t_2) - \boldsymbol{\mu}_{\mathbf{x}}(t_2)\}^{\mathsf{H}}\Big] \\ &= \mathbf{g}(t_1-t_0)\mathbf{M}\boldsymbol{\Gamma}_{\mathbf{x}_0\mathbf{x}_0}\mathbf{M}^{\mathsf{T}}\mathbf{g}^{\mathsf{H}}(t_2-t_0) + \mathbf{h}(t_1-t_0)\mathbf{M}\boldsymbol{\Gamma}_{\dot{\mathbf{x}}_0\dot{\mathbf{x}}_0}\mathbf{M}^{\mathsf{T}}\mathbf{h}^{\mathsf{H}}(t_2-t_0) + \\ &\quad \mathbf{g}(t_1-t_0)\mathbf{M}\boldsymbol{\Gamma}_{\mathbf{x}_0\dot{\mathbf{x}}_0}\mathbf{M}^{\mathsf{T}}\mathbf{h}^{\mathsf{H}}(t_2-t_0) + \mathbf{h}(t_1-t_0)\mathbf{M}\boldsymbol{\Gamma}_{\dot{\mathbf{x}}_0\mathbf{x}_0}\mathbf{M}^{\mathsf{T}}\mathbf{g}^{\mathsf{H}}(t_2-t_0) + \\ &\quad \int_{t_0}^{t_1}\int_{t_0}^{t_2} \mathbf{h}(t_2-\tau_1)\boldsymbol{\Gamma}_{\mathbf{ff}}(\tau_1,\tau_2)\mathbf{h}^{\mathsf{H}}(t_2-\tau_2)\mathrm{d}\tau_1\mathrm{d}\tau_2.\end{aligned} \tag{2.47}$$

Similar expressions can be obtained for the state-space formulation and are given by

$$\boldsymbol{\mu}_{\underline{\mathbf{x}}}(t) = \mathbb{E}[\underline{\mathbf{x}}(t)] = \underline{\mathbf{h}}(t-t_0)\mathbb{E}[\underline{\mathbf{x}}_0] + \int_{t_0}^{t} \underline{\mathbf{h}}(t-\tau)\mathbf{B}\mathbb{E}[\underline{\mathbf{f}}(\tau)]\,\mathrm{d}\tau \tag{2.48}$$

and

$$\begin{aligned}\boldsymbol{\Gamma}_{\underline{\mathbf{x}}\underline{\mathbf{x}}}(t_1,t_2) &= \mathbb{E}\Big[\left\{\underline{\mathbf{x}}(t_1)-\boldsymbol{\mu}_{\underline{\mathbf{x}}}(t_1)\right\}\left\{\underline{\mathbf{x}}(t_2)-\boldsymbol{\mu}_{\underline{\mathbf{x}}}(t_2)\right\}^{\mathsf{H}}\Big] \\ &= \underline{\mathbf{h}}(t_1-t_0)\boldsymbol{\Gamma}_{\underline{\mathbf{x}}_0\underline{\mathbf{x}}_0}\underline{\mathbf{h}}^{\mathsf{H}}(t_2-t_0)+ \\ &\qquad \int_{t_0}^{t_1}\int_{t_0}^{t_2}\underline{\mathbf{h}}(t_2-\tau_1)\mathbf{B}\boldsymbol{\Gamma}_{\underline{\mathbf{f}}\underline{\mathbf{f}}}(\tau_1,\tau_2)\mathbf{B}^{\mathsf{T}}\underline{\mathbf{h}}^{\mathsf{H}}(t_2-\tau_2)\mathrm{d}\tau_1\mathrm{d}\tau_2. \end{aligned} \tag{2.49}$$

When the input is governed by a stationary or weakly stationary random process, one can define the input random process in terms of its power spectral density, PSD, which is defined as

$$S_{FF}(\omega)=\frac{1}{2\pi}\int_{-\infty}^{\infty}e^{-\imath\omega\tau}R_{FF}(\tau)\mathrm{d}\tau \tag{2.50}$$

where $\imath=\sqrt{-1}$ and $R_{FF}(\tau)$ is the autocorrelation function of the input random process given by

$$R_{FF}(\tau)=\mathbb{E}[F(t)F(t+\tau)]\,. \tag{2.51}$$

In such cases, the PSD of the response of the system can be directly computed using a matrix triple product as

$$\mathbf{S}_{\mathbf{x}\mathbf{x}}(\omega)=\mathbf{H}(\omega)\mathbf{S}_{\mathbf{f}\mathbf{f}}(\omega)\mathbf{H}^{\mathsf{H}}(\omega),\qquad \mathbf{S}_{\underline{\mathbf{x}}\underline{\mathbf{x}}}(\omega)=\underline{\mathbf{H}}(\omega)\mathbf{S}_{\underline{\mathbf{f}}\underline{\mathbf{f}}}(\omega)\underline{\mathbf{H}}^{\mathsf{H}}(\omega) \tag{2.52}$$

where $\mathbf{H}(\omega)\in\mathbb{C}^{n\times n}$ and $\underline{\mathbf{H}}(\omega)\in\mathbb{C}^{\underline{n}\times\underline{n}}$ are obtained by substituting $s=\imath\omega$ in (2.42) and (2.45), respectively. Detailed treatments of LTI systems subjected to various kinds of random input process are available in the literature [43, 44].

2.3 Systems with Local Uncertainties and Nonlinearities

The systems of primary interest to be considered herein possess local uncertainties and/or nonlinearities and will be assumed to be of the form

$$\mathbf{M}\ddot{\mathbf{x}}(t)+\mathbf{D}\dot{\mathbf{x}}(t)+\mathbf{K}\mathbf{x}(t)+\mathbf{L}\mathbf{u}(\mathbf{x},\dot{\mathbf{x}},\mathbf{p}_u,t)=\mathbf{f}(t),\quad \mathbf{x}(t_0)=\mathbf{x}_0,\dot{\mathbf{x}}(0)=\dot{\mathbf{x}}_0 \tag{2.53}$$

where $\mathbf{u}(\mathbf{x},\dot{\mathbf{x}},\mathbf{p}_u,t)\in\mathbb{R}^{N_p}$ is a nonlinear vector function of the states of the system that prescribes the uncertainties and the nonlinearities of the system; $\mathbf{L}\in\mathbb{R}^{n\times N_p}$ is an influence matrix that maps the uncertainties and the nonlinearities to appropriate degrees-of-freedom of the system; $\mathbf{p}_u$ represents the uncertain parameters on which the

function $\mathbf{u}$ depends. It is evident that only $N_p \ll n$ degrees-of-freedom, and not all, are involved in either uncertain or nonlinear relationships.

A similar form of the system can be assumed for the state-space representation given by

$$\dot{\underline{\mathbf{x}}}(t) = \mathbf{A}\underline{\mathbf{x}}(t) + \mathbf{B}\underline{\mathbf{f}}(t) + \mathbf{L}\underline{\mathbf{u}}(\underline{\mathbf{x}}, \mathbf{p}_u, t), \quad \underline{\mathbf{x}}(t_0) = \underline{\mathbf{x}}_0 \tag{2.54}$$

where $\underline{\mathbf{u}}(\underline{\mathbf{x}}, \mathbf{p}_u, t) \in \mathbb{R}^{N_p}$ and $\mathbf{L} \in \mathbb{R}^{\underline{n} \times N_p}$ are as defined earlier.

The presence of the additional nonlinear and/or random vector function in the system's equations of motion can affect its stability. The stability of dynamical systems is discussed in the next section. Moreover, it is extremely challenging, if not impossible, to find closed-form analytical solutions for such systems. This necessitates the use of numerical methods, which will also be discussed later in this chapter.

When the system has both uncertainties and/or nonlinearities, it is possible to compute its response utilizing the principle of superposition due to the form of the equations of motion of the system in (2.54). The equations of motion can be viewed as a linear system in (2.2) subjected to a nonlinear forcing term given by $\mathbf{L}\underline{\mathbf{u}}$. Therefore, the solution of (2.54) can be computed by first computing the response of the linear portion of the system, given by the convolution integral in (2.30) and then adding the nonlinear portion as

$$\underline{\mathbf{x}}(t) = \underline{\mathbf{h}}(t - t_0)\underline{\mathbf{x}}_0 + \int_{t_0}^{t} \underline{\mathbf{h}}(t - \tau)\mathbf{B}\underline{\mathbf{f}}(\tau)\mathrm{d}\tau + \int_{t_0}^{t} \underline{\mathbf{h}}_L(t - \tau)\underline{\mathbf{u}}(\underline{\mathbf{x}}, \dot{\underline{\mathbf{x}}}, \mathbf{p}_u, \tau)\mathrm{d}\tau. \tag{2.55}$$

In (2.55), the first two terms represent the response of the linear system, (2.2). The last term represents the response of a dynamical system subjected to a load given by $\mathbf{L}\underline{\mathbf{u}}$ and initial conditions given by the response of the linear portion of the system. $\underline{\mathbf{h}}_L(t)$ is the impulse response function of the linear system, (2.2), subjected to a unit impulse in the pattern of the uncertainties and the nonlinearities. If $\delta(t)$ is the Dirac's delta function, $\underline{\mathbf{h}}_L(t)$ is the response of the following system of ODEs:

$$\dot{\underline{\mathbf{h}}}_L(t) = \mathbf{A}\underline{\mathbf{h}}_L(t) + \mathbf{L}\delta(t). \tag{2.56}$$

Further, (2.55) can be recognized as a nonlinear Volterra integral equation (NVIE) of the second kind.

2.4 Stability of Dynamical Systems

The stability theory of LTI systems is well studied [182, 183]. If a given LTI system is stable, the stability criterion usually prescribes an upper bound on how much the system can be perturbed while it still remains stable. The types of stability that can characterize a dynamical system's behavior are given in Table 2.1, with reference to the state-space representation of the system.

Table 2.1: Stability of Dynamical Systems

Stability type	**General definition**	**Characteristics of linear system**
Lyapunov stable	$\forall \underline{\mathbf{x}}_0$, $\exists \epsilon > 0$, such that $\forall t \geq 0$, $\|\underline{\mathbf{x}}(t)\| < \epsilon$	All eigenvalues of $\mathbf{A}$ lie in the closed left half plane and every eigenvalue with a zero real part is semi-simple
Asymptotically stable	$\forall \underline{\mathbf{x}}_0$, $\lim_{t\to\infty} \underline{\mathbf{x}}(t) = 0$	All eigenvalues of $\mathbf{A}$ lie in the open left half plane, *i.e.*, all eigenvalues have negative real parts
Unstable	Not Lyapunov stable	An eigenvalue of $\mathbf{A}$ lie in the open right half plane, *i.e.*, at least one eigenvalue has positive real part

The subject of stability of dynamical systems with uncertainties and nonlinearities is a vast research area in itself and will not be discussed in detail herein. Interested readers can refer to [184–187] and the references therein. In the subsequent chapters, the stability of pertinent systems will be discussed individually as required.

2.5 Numerical Methods to Analyze Dynamical Systems

The numerical methods developed to solve dynamical systems can be viewed as special cases of the general theory of numerical methods for PDEs, ODEs and IEs. The discussion herein will be restricted to numerical methods for ODEs and NVIEs. Numerical methods for ODEs include the finite difference method, collocation methods, multi-stage time-stepping methods (*e.g.* Rünge-Kutta method [188, 189]). Butcher [190] provides an excellent in-depth analysis of these methods. With the exception of

finite difference methods, all of the aforementioned methods involve first-order ODEs. Methods have also been developed to directly address second-order ODEs as they appear frequently in structural dynamics applications. These methods include Newmark's beta method [191] and Wilson's theta method [192]. Bert [193] provides a comparison of several numerical integration techniques in the context of nonlinear dynamical systems.

Numerical methods for NVIEs build upon numerical integration techniques similar to those utilized for ODEs. Detailed treatments of numerical methods for linear and nonlinear Volterra integral equations can be found in the literature [139, 141, 194]. The following subsections present details of the numerical techniques relevant to the content of this dissertation.

2.5.1 Rünge-Kutta Methods for ODEs

Consider a system of first-order ODEs given by

$$\dot{\mathbf{y}}(t) = \mathbf{f}\left(\mathbf{y}(t), t\right); \quad \mathbf{y}(0) = \mathbf{y}_0. \tag{2.57}$$

The existence and uniqueness of the solution to (2.57) is ensured if $\mathbf{f}$ is Lipschitz continuous in $\mathbf{y}$ and continuous in t [195].

An m-stage Rünge-Kutta method to solve (2.57) can be represented using the Butcher tableau [190] given as

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^{\mathsf{T}} \end{array} \tag{2.58}$$

where $\mathbf{c} = \{c_i\}_{i=1}^m$ relates the position of the stage values, *i.e.*, discretization of t, $\mathbf{A} = \{A_{i,j}\}_{i,j=1}^m$ relates the dependence of the stages on the derivatives found at other stages, and $\mathbf{b} = \{b_i\}_{i=1}^m$ is essentially a vector of quadrature weights.

The update equation and the stages to solve (2.57) can be written as

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \sum_{i=1}^{m} b_i \mathbf{k}_i, \quad n = 0, 1, 2, \ldots, N_t - 1 \tag{2.59}$$

$$\mathbf{k}_i = \mathbf{f}\left(\mathbf{y}_n + \Delta t \sum_{j=1}^{m} A_{i,j} \mathbf{k}_j, t_n + c_i \Delta t\right) \tag{2.60}$$

where Δt is the increment in the value of t at each step, $\mathbf{y}_n = \mathbf{y}(t_n)$ and N_t is the total number of points in the discretization of t. The family of Rünge-Kutta methods can be classified based on the structure of the matrix $\mathbf{A}$ as follows:

- Explicit: if $A_{i,j} = 0 \; \forall j \geq i$.
- Diagonally implicit: if $A_{i,j} = 0 \; \forall j > i$ and $A_{i,j} \neq 0$ for some $j = i$.
- Implicit: if $A_{i,j} \neq 0$ for some $j > i$.

The global (cumulative) error associate with an explicit fourth-order Rünge-Kutta method which will be employed later in the dissertation is

$$|\mathbf{y}(t_n) - \mathbf{y}_n| \leq C\Delta t^4 \tag{2.61}$$

where C is some constant. A detailed discussion on the error behavior and the numerical stability of the family of Rünge-Kutta methods can be found in [190].

2.5.2 Numerical Methods for NVIEs

Consider a general nonlinear Volterra integral equation of the second kind given by

$$\mathbf{y}(t) = \mathbf{x}(t) + \int_0^t \mathbf{f}\left(\mathbf{y}(\tau), t, \tau\right) \mathrm{d}\tau \tag{2.62}$$

where $\mathbf{x}(t)$ and $\mathbf{f}\left(\mathbf{y}(\tau), t, \tau\right)$ are known functions and $\mathbf{y}(t)$ is the unknown function to be determined; $\mathbf{f}$ is oftentimes referred to as the Volterra kernel. (2.62) possesses a unique solution if $\mathbf{f}$ is Lipschitz continuous in $\mathbf{y}$ and continuous in t, τ and $\mathbf{x}$ is continuous in t [139].

One method to solve such equations is to discretize (2.62) using a numerical quadrature rule as

$$\mathbf{y}_n = \mathbf{x}_n + \Delta t \sum_{i=0}^{n} w_{n,i} \mathbf{f}\left(\mathbf{y}_i, t_n, t_i\right), \quad n = n_0, \ldots, N_t \tag{2.63}$$

where Δt is again the increment in the value of t at each step, $\mathbf{y}_n = \mathbf{y}(t_n)$, $\mathbf{x}_n = \mathbf{x}(t_n)$ and N_t is the total number of points in the discretization of t. $w_{n,i}$ are the quadrature weights and t_i's are usually computed from a pre-defined set of quadrature abscissas. It should be noted that in (2.63), n may start from a value $n_0 \neq 0$ as certain quadrature rules require a minimum number of points, n_0, before they can be applied. In such cases, the first n_0 points must be computed using some other method, *e.g.* a matching-order Rünge-Kutta method. The application of the Rünge-Kutta method to solve NVIEs

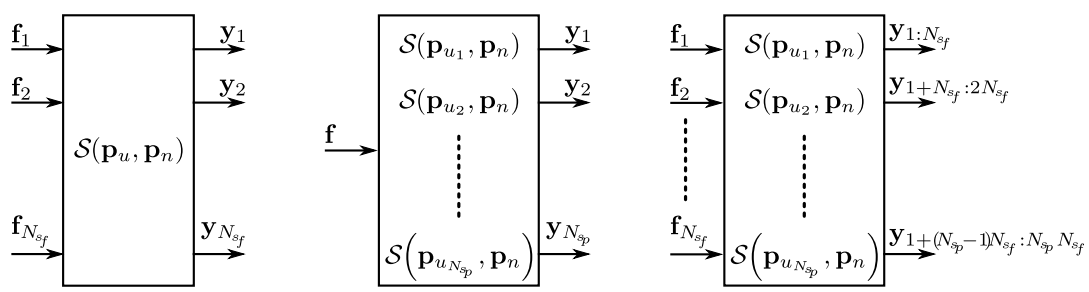
along with a detailed discussion of the error behavior and the stability of the numerical methods utilized to solve such NVIEs can be found in the literature [139]. The global (cumulative) error associate with the use of a quadrature-based method in case of linear Volterra kernels, which will be employed later in the dissertation is

$$|\mathbf{y}(t_n) - \mathbf{y}_n| \leq C\Delta t^p \tag{2.64}$$

where p is the order of the numerical quadrature used.

2.6 Sampling-based Methods for Uncertainty Quantification

This section will discuss the application of sampling-based methods for the determination of relevant statistical properties of the response of a system with load and parameter uncertainties. Sampling-based methods function by first generating samples of the input uncertain parameters, then employing a deterministic solver to compute corresponding samples of the system's response and finally computing the relevant response statistics and probabilities from these samples. Figure 2.6 illustrates



(a) Load only (b) Parametric only (c) Both load and parametric

Figure 2.1: Sampling-based Approach for Uncertainty Quantification

the functioning of sampling-based methods in the context of the conceptual model presented in Figure 1.2. N_{s_f} and N_{s_p} represent the number of samples of the random load, $\mathbf{f}$, and the uncertain parameters, $\mathbf{p}_u$, respectively. Once the samples have been computed, the relevant statistics can be computed via standard statistical techniques.

For instance, the mean and cross-covariance functions of the response processes can be obtained using the following unbiased estimators:

$$\boldsymbol{\mu}_{\mathbf{y}}(t) = \frac{1}{N}\sum_{i=1}^{N}\mathbf{y}_i(t) \tag{2.65}$$

and

$$\boldsymbol{\Gamma}_{\mathbf{yy}}(t_1, t_2) = \frac{1}{N-1}\sum_{i=1}^{N}\left(\mathbf{y}_i(t_1) - \boldsymbol{\mu}_{\mathbf{y}}(t_1)\right)\left(\mathbf{y}_i(t_2) - \boldsymbol{\mu}_{\mathbf{y}}(t_2)\right)^{\mathsf{T}} \tag{2.66}$$

where

$$N = \begin{cases} N_{s_f} & \text{for load uncertainty only} \\ N_{s_p} & \text{for parametric uncertainty only} \\ N_{s_f}N_{s_p} & \text{for both load and parametric uncertainty} \end{cases} \tag{2.67}$$

and $\mathbf{y}_i$ denotes the ith sample of the system's output. A variety of techniques can be utilized for the generation of sample points, as discussed in Section 1.1.

2.7 Computational Performance of Algorithms

This section discusses the computational costs associated with common mathematical operations, when implemented sequentially, that will be utilized in subsequent chapters to quantify the computational efficiency of the developed methods. It should be noted that the mathematical symbols utilized in this section are unrelated to those in the rest of the dissertation. A given sequential algorithm can be evaluated in terms of its execution time, expressed as a function of the size of its input. Another metric that can be utilized as a performance metric of a given sequential algorithm is its FLOP count, *i.e.*, the number of floating point operations that must be performed during its execution. While both these metrics can be utilized individually and interchangeably, their combined usage enables the following important insights regarding the implementation of a given algorithm:

- performance comparison of different implementations of a given algorithm on the same machine
- performance comparison of the same implementation of a given algorithm on different machines

- optimality of a given implementation of a given algorithm

The computational complexities (asymptotic FLOP counts) of relevant algorithms are presented in the remainder of this subsection. Table 2.2 presents the asymptotic computational cost of some common matrix operations. A detailed analysis of a variety of matrix algorithms can be found in the literature [196].

Table 2.2: Computational Complexity of Matrix Algorithms

Operation	**Complexity**
Matrix-matrix product ($m\times n$ by $n\times p$)	mnp
Matrix-vector product ($m\times n$ by $n\times 1$)	mn
Matrix inverse ($n\times n$, via LU decomposition)	$2n^3/3$
Eigenvalue decomposition (symmetric $n\times n$, via QR decomposition)	$4n^3/3$
Eigenvalue decomposition (non-symmetric $n\times n$, via QZ decomposition)	$10n^3/3$
Cholesky factorization (symmetric $n\times n$)	$n^3/3$

It should be noted that the computational costs presented in Table 2.2 are costs when the relevant matrix operations are performed on real matrices. For complex matrices, the costs are multiplied by a factor of 4. Moreover, the costs presented are asymptotic costs, *i.e.*, lower order terms in the complexity analysis have been neglected.

The computational complexities of some other relevant algorithms are as follows:

- Fast Fourier transform (FFT): of a vector of length n, using radix-2 algorithm
  - $n \log_2 n$

- Vector convolution: of two vectors of length n
  - direct summation: n^2
  - using FFT: $n \log_2 n$

- Rünge-Kutta method: explicit, m-stage method to solve a linear system of N_s ODEs and N_t points in the discretized domain
  - $mN_tN_s^2$

This chapter summarized the theoretical background related to the analysis of linear dynamical systems required to facilitate the discussion in the remainder of the

dissertation. It also presented the general representation of systems with local features, uncertainties and nonlinearities, which are the focus of the analysis methods developed in later chapters. In addition, a brief overview of numerical methods employed to analyze systems of interest and sampling-based uncertainty quantification techniques was also presented. The next three chapters will discuss the development of efficient computational techniques to perform the modal analysis, frequency domain analysis and time domain analysis of systems with local features, respectively. The final chapter will discuss the use of GPUs for uncertainty quantification of computational models.

Chapter 3

Modal Response of Systems

This chapter discusses the development of an efficient computational methodology for the modal analysis of undamped linear systems with local stiffness uncertainties. The newly developed method utilizes an enriched basis that consists of the sub-spectrum of a nominal system augmented with additional basis vectors generated from a knowledge of the structure of the stiffness uncertainty.

The free vibration of an n-dimensional undamped, deterministic linear dynamical system can be represented as

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{K}\mathbf{x}(t) = \mathbf{0}_{n\times 1} \tag{3.1}$$

where $\mathbf{M}$ and $\mathbf{K} \in \mathbb{R}^{n\times n}$ represent the mass and the stiffness matrices of the deterministic system, respectively. The system given by (3.1) will be referred to as the 'nominal' system in the remainder of this chapter. It is also assumed that $\mathbf{M}$ is a symmetric, positive definite matrix and $\mathbf{K}$ is a symmetric, positive semi-definite matrix. Following the discussion in Section 2.1.1, the generalized eigenvalue problem governing the free vibration of the nominal system can be written as

$$\mathbf{K}\boldsymbol{\Phi} = \mathbf{M}\boldsymbol{\Phi}\boldsymbol{\Lambda} \tag{3.2}$$

where $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times n}$ is a diagonal matrix consisting of the eigenvalues of the system, and $\boldsymbol{\Phi} \in \mathbb{R}^{n\times n}$ is the modal matrix consisting of mass-normalized eigenvectors of the system

given by

$$\mathbf{\Lambda} = \mathrm{diag}\big[\lambda_1, \lambda_2, \ldots, \lambda_n\big]\,, \quad \mathbf{\Phi} = \begin{bmatrix} \phi_{1,1} & \phi_{2,1} & \cdots & \phi_{n,1} \\ \phi_{1,2} & \phi_{2,2} & \cdots & \phi_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{1,n} & \phi_{2,n} & \cdots & \phi_{n,n} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 & \ldots \boldsymbol{\phi}_n \end{bmatrix} \tag{3.3}$$

such that

$$\mathbf{\Phi}^{\mathsf{T}}\mathbf{M}\mathbf{\Phi} = \mathbf{I}_{n\times n} \tag{3.4}$$

$$\mathbf{\Phi}^{\mathsf{T}}\mathbf{K}\mathbf{\Phi} = \mathbf{\Lambda}. \tag{3.5}$$

The free vibration of an n-dimensional linear dynamical system with stiffness uncertainties can be represented in a form similar to (3.1) as

$$\mathbf{M}\ddot{\mathbf{v}}(t) + \mathbf{K}_u\mathbf{v}(t) = \mathbf{0}_{n\times 1} \tag{3.6}$$

where the mass matrix, $\mathbf{M}$, is same as that in (3.1) and the random stiffness matrix, $\mathbf{K}_u$, can be represented as a perturbed form of the nominal stiffness matrix, $\mathbf{K}$, in (3.1) as

$$\mathbf{K}_u = \mathbf{K} + \Delta\mathbf{K} \tag{3.7}$$

where $\Delta\mathbf{K} \in \mathbb{R}^{n\times n}$ is a symmetric matrix of rank N_p containing the uncertainties. For the systems of interest herein, it will be assumed that $N_p \ll n$, *i.e.*, the uncertainties are spatially localized in the system. In such cases, $\Delta\mathbf{K}$ will be a relatively sparse matrix and can be represented without loss of generality as

$$\Delta\mathbf{K} = \mathbf{L}\delta\mathbf{K}\mathbf{L}^{\mathsf{T}} \tag{3.8}$$

where $\delta\mathbf{K} \in \mathbb{R}^{N_p\times N_p}$ is a dense matrix containing the uncertainties in the stiffness of the system and $\mathbf{L} \in \mathbb{R}^{n\times N_p}$ denotes the positions of the uncertainties of the system. The matrix $\mathbf{L}$ is oftentimes a permutation of an $N_p\times N_p$ identity matrix combined with appropriate zero matrices.

The next section will discuss a conventional method to compute the relevant eigenpair statistics which will be called the 'brute-force method' in the remainder of the chapter. The subsequent section will develop a new approach proposed for the modal analysis of linear systems with local stiffness uncertainties. Finally, numerical examples will be presented to demonstrate the computational efficiency of the proposed method as compared to that of the brute-force method.

3.1 Brute-Force Method

The generalized eigenvalue problem associated with the uncertain system in (3.6) can be stated as

$$\mathbf{K}_u \mathbf{\Psi} = \mathbf{M} \mathbf{\Psi} \mathbf{\Sigma} \tag{3.9}$$

where $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ is a diagonal matrix consisting of the eigenvalues of the system, and $\mathbf{\Psi} \in \mathbb{R}^{n \times n}$ is the modal matrix consisting of mass-normalized eigenvectors of the system given by

$$\mathbf{\Sigma} = \mathrm{diag}\big[\sigma_1, \sigma_2, \ldots, \sigma_n\big], \quad \mathbf{\Psi} = \begin{bmatrix} \psi_{1,1} & \psi_{2,1} & \cdots & \psi_{n,1} \\ \psi_{1,2} & \psi_{2,2} & \cdots & \psi_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{1,n} & \psi_{2,n} & \cdots & \psi_{n,n} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\psi}_1 & \boldsymbol{\psi}_2 & \ldots \boldsymbol{\psi}_n \end{bmatrix} \tag{3.10}$$

such that

$$\mathbf{\Psi}^{\mathsf{T}} \mathbf{M} \mathbf{\Phi} = \mathbf{I}_{n \times n} \tag{3.11}$$

$$\mathbf{\Psi}^{\mathsf{T}} \mathbf{K}_u \mathbf{\Phi} = \mathbf{\Sigma}. \tag{3.12}$$

Let it be assumed that the eigenvalues and the eigenvectors in (3.10) are written in the increasing order of the magnitudes of the eigenvalues, *i.e.*, $\sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_n$ and that only the first N_o smallest eigenvalues and corresponding eigenvectors of the uncertain system are required. These desired eigenvalues and eigenvectors can then be written as

$$\mathbf{\Sigma}_1 = \mathrm{diag}\big[\sigma_1, \sigma_2, \ldots, \sigma_{N_o}\big], \quad \mathbf{\Psi}_1 = \begin{bmatrix} \boldsymbol{\psi}_1 & \boldsymbol{\psi}_2 & \ldots \boldsymbol{\psi}_{N_o} \end{bmatrix} \tag{3.13}$$

such that

$$\mathbf{\Psi}_1^{\mathsf{T}} \mathbf{M} \mathbf{\Psi}_1 = \mathbf{I}_{N_o \times N_o} \tag{3.14}$$

$$\mathbf{\Psi}_1^{\mathsf{T}} \mathbf{K}_u \mathbf{\Psi}_1 = \mathbf{\Sigma}_1. \tag{3.15}$$

These eigenvalues and the eigenvectors will be random in nature because of the uncertainty of $\mathbf{K}_u$. Employing a sampling-based approach, the mean and the variance of the eigenvalues and eigenvectors can be computed by utilizing the following unbiased

estimators:

$$\mu_{\sigma_i} = \frac{1}{N_s}\sum_{j=1}^{N_s}\sigma_i^{(j)} \tag{3.16}$$

$$\Gamma_{\sigma_i\sigma_i} = \frac{1}{N_s-1}\sum_{j=1}^{N_s}(\sigma_i^{(j)}-\mu_{\sigma_i})^2 \tag{3.17}$$

$$\boldsymbol{\mu}_{\boldsymbol{\psi}_i} = \frac{1}{N_s}\sum_{j=1}^{N_s}\boldsymbol{\psi}_i^{(j)} \tag{3.18}$$

$$\boldsymbol{\Gamma}_{\boldsymbol{\psi}_i\boldsymbol{\psi}_i} = \frac{1}{N_s-1}\sum_{j=1}^{N_s}\left[(\boldsymbol{\psi}_i^{(j)}-\boldsymbol{\mu}_{\boldsymbol{\psi}_i})(\boldsymbol{\psi}_i^{(j)}-\boldsymbol{\mu}_{\boldsymbol{\psi}_i})^{\mathsf{T}}\right] \tag{3.19}$$

where N_s is the number of samples computed and $i = 1, 2, \ldots, N_o$. Algorithm 1 outlines the brute-force procedure to compute the statistics of eigenproperties of the uncertain system. The computational cost associated with a particular step is listed in parentheses. Thus, the total computational cost associated with computing all the samples of the

Algorithm 1 Brute-Force Method to Compute $\boldsymbol{\Psi}_1$ and $\boldsymbol{\Sigma}_1$

Require: $\mathbf{K}$, $\mathbf{M}$

1: Generate N_s samples of the stiffness uncertainties, $\Delta\mathbf{K}$
2: **for all** samples of $\Delta\mathbf{K}$ **do**
3: Compute $\mathbf{K}_u$ using (3.7) $(-)$
4: Compute $\boldsymbol{\Psi}_1$ and $\boldsymbol{\Sigma}_1$ using (3.9) $(4n^3/3)$
5: **end for**
6: Compute required statistics using (3.16)–(3.19)

first N_o eigenpairs of the uncertain system using the brute-force method will be

$$C_b = \frac{4}{3}n^3 N_s \tag{3.20}$$

It should be noted that the computational cost associated with Step 4 of Algorithm 1 has been assumed to be $4n^3/3$ which corresponds to the computation of all the eigenpairs using the symmetric QR factorization of $\mathbf{K}_u$. Since only a subset of the eigenpairs is required and the matrix $\mathbf{K}_u$ is usually sparse, more efficient algorithms can be utilized to compute the relevant eigenpairs. In the implementation, Matlab's `eigs` function has been utilized to compute the relevant eigenpairs; `eigs` first reduces the given

matrix in a symmetric tridiagonal form using the Lanczos algorithm (Algorithm 9.2.1 in [196]) and then computes the relevant eigenpairs using the symmetric QR algorithm (Algorithm 8.3.3 in [196]).

The computational cost of the Lanczos algorithm is approximately $(2n_z + 8)nn_s$ where n_z is the average number of nonzero entries in each row of $\mathbf{K}_u$ and n_s is the number of Lanczos steps performed. The cost of the symmetric QR algorithm applied to a symmetric tridiagonal matrix resulting from the Lanczos algorithm is n. Thus, the total computational cost associated with computing all the samples of the first N_o eigenpairs of the uncertain system using the brute-force method will be

$$C_b^* = (n + n_s(2n_z + 8)n)\, N_o N_s. \tag{3.21}$$

Thus, the effective computational cost of the implementation of `eigs` for a relatively dense matrix ($n_z \approx n$) is between $\mathcal{O}\left(n^2\right)$ and $\mathcal{O}\left(n^3\right)$; this computational cost greatly depends upon the number of Lanczos steps, n_s, which increases rapidly as the number of eigenpairs required to be computed, N_o, increases.

3.2 Proposed Method

Let it also be assumed that the eigenvectors and the eigenvalues of the nominal system be also sorted in increasing order of the magnitude of the eigenvalues and that the smallest N_o eigenvalues and corresponding eigenvectors are given by

$$\mathbf{\Lambda}_1 = \mathrm{diag}\big[\lambda_1, \lambda_2, \ldots, \lambda_{N_o}\big]\,, \quad \mathbf{\Phi}_1 = \big[\boldsymbol{\phi}_1 \quad \boldsymbol{\phi}_2 \quad \ldots \boldsymbol{\phi}_{N_o}\big] \tag{3.22}$$

so that

$$\mathbf{\Phi}_1^{\mathsf{T}} \mathbf{M} \mathbf{\Phi}_1 = \mathbf{I}_{N_o \times N_o} \tag{3.23}$$

$$\mathbf{\Phi}_1^{\mathsf{T}} \mathbf{K} \mathbf{\Phi}_1 = \mathbf{\Lambda}_1. \tag{3.24}$$

Since the mass matrix is assumed to be deterministic, it is advantageous to rewrite the equations of motion of the nominal and the uncertain system as

$$\ddot{\widehat{\mathbf{x}}}(t) + \widehat{\mathbf{K}}\widehat{\mathbf{x}}(t) = \mathbf{0}_{n\times 1} \tag{3.25}$$

$$\ddot{\widehat{\mathbf{v}}}(t) + \widehat{\mathbf{K}}_u\widehat{\mathbf{v}}(t) = \mathbf{0}_{n\times 1} \tag{3.26}$$

where

$$\widehat{\mathbf{x}}(t) = \mathbf{M}^{1/2}\mathbf{x}(t) \tag{3.27}$$

$$\widehat{\mathbf{v}}(t) = \mathbf{M}^{1/2}\mathbf{v}(t) \tag{3.28}$$

$$\widehat{\mathbf{K}} = \mathbf{M}^{-1/2}\mathbf{K}\mathbf{M}^{-1/2} \tag{3.29}$$

$$\widehat{\mathbf{K}}_u = \mathbf{M}^{-1/2}\mathbf{K}_u\mathbf{M}^{-1/2}. \tag{3.30}$$

$\mathbf{M}^{1/2}$ is the principal square root of $\mathbf{M}$, *i.e.*, $\mathbf{M} = \mathbf{M}^{1/2}\mathbf{M}^{1/2}$. (3.30) can be rewritten using (3.7) and (3.8) as

$$\widehat{\mathbf{K}}_u = \widehat{\mathbf{K}} + \widehat{\Delta\mathbf{K}} \tag{3.31}$$

where

$$\widehat{\Delta\mathbf{K}} = \widehat{\mathbf{L}}\delta\mathbf{K}\widehat{\mathbf{L}}^{\mathsf{T}} \tag{3.32}$$

$$\widehat{\mathbf{L}} = \mathbf{M}^{-1/2}\mathbf{L}. \tag{3.33}$$

The generalized eigenvalue problems, (3.2) and (3.9), can now be reposed as standard, symmetric eigenvalue problems as

$$\widehat{\mathbf{K}}\widehat{\mathbf{\Phi}} = \widehat{\mathbf{\Phi}}\mathbf{\Lambda} \tag{3.34}$$

$$\widehat{\mathbf{K}}_u\widehat{\mathbf{\Psi}} = \widehat{\mathbf{\Psi}}\mathbf{\Sigma}. \tag{3.35}$$

As a result of this transformation, the eigenvalues remain unchanged while the eigenvectors of the standard eigenvalue problems, (3.34) and (3.35), are related to those of the generalized eigenvalue problems, (3.2) and (3.9), by

$$\widehat{\mathbf{\Phi}} = \mathbf{M}^{1/2}\mathbf{\Phi} \tag{3.36}$$

$$\widehat{\mathbf{\Psi}} = \mathbf{M}^{1/2}\mathbf{\Psi}. \tag{3.37}$$

Specifically, the first N_o eigenvectors of the transformed nominal and uncertain systems, $\widehat{\mathbf{\Phi}}_1$ and $\widehat{\mathbf{\Psi}}_1$, can be written in terms of the first N_o eigenvectors of the original nominal and uncertain systems as

$$\widehat{\mathbf{\Phi}}_1 = \mathbf{M}^{1/2}\mathbf{\Phi}_1 \tag{3.38}$$

$$\widehat{\mathbf{\Psi}}_1 = \mathbf{M}^{1/2}\mathbf{\Psi}_1. \tag{3.39}$$

In the proposed method, the n-dimensional standard eigenvalue problem in (3.35) is transformed into an $N_r < n$ dimensional standard eigenvalue problem by first approximating the eigenvectors using a reduced dimensional space of N_r generalized coordinates and then computing the relevant eigenvalues by solving an eigenvalue problem in the reduced space. The Ritz method is utilized to approximate $\widehat{\mathbf{\Psi}}_1$ in an N_r-dimensional space as

$$\widehat{\mathbf{\Psi}}_1 \approx \mathbf{TQ} \tag{3.40}$$

where $\mathbf{T} \in \mathbb{R}^{n \times N_r}$ is a transformation matrix whose columns are the orthonormal basis vectors of the approximation and $\mathbf{Q} \in \mathbb{R}^{N_r \times N_r}$ is a matrix whose columns are elements of the N_r-dimensional generalized coordinate space. The accuracy and the computational efficiency of the Ritz approximation hinges on a judicious selection of the approximation basis vectors, $\mathbf{T}$. Most previously proposed eigen-reanalysis methods are not directly applicable to the problems of interest targeted here. Ideally, one would like to construct a single Ritz basis space that incorporates the knowledge of the structure of the uncertain perturbation, $\widehat{\Delta\mathbf{K}}$, or equivalently, $\Delta\mathbf{K}$, and the knowledge that the perturbation has a much smaller rank than the size of the nominal system matrices.

As discussed in [197], there are several natural choices for the Ritz basis. One obvious choice of the Ritz basis vectors is the first N_o eigenvectors of the transformed nominal system, *i.e.*,

$$\mathbf{T} = \widehat{\mathbf{\Phi}}_1. \tag{3.41}$$

Large errors can be encountered when (3.41) is employed as the Ritz basis, especially when the perturbation, $\widehat{\Delta\mathbf{K}}$, possesses large magnitude elements or when it causes a fundamental change in the nature of the system. Alternatively, if one assumes that the perturbation, $\widehat{\Delta\mathbf{K}}$, is a function of a vector of uncertain parameters, $\mathbf{p}_u$, another possible Ritz basis is the augmentation of the first N_o eigenvectors of the transformed nominal system with their sensitivities to this uncertain parameter vector, *i.e.*,

$$\mathbf{T} = \text{span}\begin{bmatrix}\widehat{\mathbf{\Phi}}_1 & \frac{\partial \widehat{\mathbf{\Phi}}_1}{\partial \mathbf{p}_u}\end{bmatrix} \tag{3.42}$$

where span[...] denotes an orthogonal basis for the linear vector space spanned by its argument. The enrichment given by (3.42) can improve the accuracy of the approximation; however, it is well known that eigenvector sensitivities are often

numerically ill-conditioned [198] thus limiting the utility of this choice of $\mathbf{T}$. Another possible approach, termed the multi-model approach [199], constructs a basis that consists of the system eigenvectors at a collection of parameter realizations, *i.e.*,

$$\mathbf{T} = \text{span}\left[\widehat{\mathbf{\Phi}}_1 \quad \widehat{\mathbf{\Psi}}_1|_{\mathbf{p}_u=\mathbf{p}_{u_1}} \quad \widehat{\mathbf{\Psi}}_1|_{\mathbf{p}_u=\mathbf{p}_{u_1}} \quad \ldots \quad \widehat{\mathbf{\Psi}}_1|_{\mathbf{p}_u=\mathbf{p}_{u_j}}\right]. \tag{3.43}$$

While this is an attractive choice for the basis in some applications, it suffers from the difficulty of how to efficiently select the most important parameter points at which to evaluate the model. Instead of incorporating the numerical values of the perturbation, alternatively, one can construct a basis that incorporates the structure of the perturbation into the construction process. To motivate the selection of the approximation basis used herein, one can recast (3.26) in the following form:

$$\ddot{\widehat{\mathbf{v}}}(t) + \widehat{\mathbf{K}}\widehat{\mathbf{v}}(t) = -\widehat{\Delta\mathbf{K}}\widehat{\mathbf{v}}(t) \tag{3.44}$$

or

$$\ddot{\widehat{\mathbf{v}}}(t) + \widehat{\mathbf{K}}\widehat{\mathbf{v}}(t) = \widehat{\mathbf{L}}\mathbf{f}(t) \tag{3.45}$$

where $\mathbf{f}(t) = -\delta\mathbf{K}\widehat{\mathbf{L}}^{\mathsf{T}}\widehat{\mathbf{v}}(t)$.

Although the pseudo-force, $\mathbf{f}(t)$, caused by the stiffness matrix perturbation is not known explicitly, it can be noted that the excitation imparted on the system in (3.45) will always lie in the column space of $\widehat{\mathbf{L}}$. Previously, algorithms have been developed for the efficient forced response analysis of systems of the form (3.45) which incorporate the pattern of loading, $\widehat{\mathbf{L}}$, into the construction of the basis. In [110], the term load-dependent Ritz vector (LDRV) was coined and shown to be the elements of a block Krylov sequence arising from $\widehat{\mathbf{K}}^{-1}$ and $\widehat{\mathbf{L}}$. Kline [200] utilized a combination of exact eigenvectors and these so-called LDRVs. A similar approach was utilized by Chu and Milman [201] to investigate the effects of an added viscous point connected absorber on the damped natural frequencies of linear systems. Given a matrix, $\widehat{\mathbf{K}}$, a set of starting vectors given by $\widehat{\mathbf{L}}$, (3.32), and the number of blocks, N_b (size of the block Krylov space), to be used in the enrichment process, a sequence of $n \times N_r$ matrices can be defined as

$$\left[\widehat{\mathbf{K}}^{-1}\widehat{\mathbf{L}}, \quad \widehat{\mathbf{K}}^{-2}\widehat{\mathbf{L}}, \quad \ldots, \quad \widehat{\mathbf{K}}^{-N_b}\widehat{\mathbf{L}}\right]. \tag{3.46}$$

The basis to be used herein is constructed by augmenting the eigenvectors of the transformed nominal system, $\widehat{\mathbf{\Phi}}_1$, with elements of the sequence of matrices given by

(3.46) and can be written as

$$\mathbf{T} = \text{span}\left[\widehat{\mathbf{\Phi}}_1, \quad \widehat{\mathbf{K}}^{-1}\widehat{\mathbf{L}}, \quad \widehat{\mathbf{K}}^{-2}\widehat{\mathbf{L}}, \quad \ldots, \quad \widehat{\mathbf{K}}^{-N_b}\widehat{\mathbf{L}}\right]. \tag{3.47}$$

The first block of enrichment vectors given by the sequence in (3.46) represents the response of the system to a static load in the pattern of $\widehat{\mathbf{L}}$, equivalent to the static correction term encountered in the mode-acceleration method. The subsequent blocks in the sequence represent additional enrichment terms due to inertia effects of the dynamical system. The contribution of the method developed in this chapter is not in the generation of these enrichment vectors but rather the observation that the effects of fixed structure random perturbations can be recast into this form. It should also be noted that the basis proposed here differs from that used in the combined approximation method for eigen-reanalysis [99–102] in two significant ways. First, a single approximation basis is generated once for all perturbations, rather than repeating the approximation for every perturbation, and second, the sub-spectrum, *i.e.*, the first N_o-eigenpairs, are calculated simultaneously rather than one at a time.

Once the approximation basis, $\mathbf{T}$ has been constructed, the n-dimensional standard eigenvalue problem in (3.35) can be reduced to an N_r-dimensional eigenvalue problem as

$$\widehat{\mathbf{K}}_{u_r}\widetilde{\mathbf{Q}} = \widetilde{\mathbf{Q}}\widetilde{\mathbf{\Sigma}} \tag{3.48}$$

where

$$\widehat{\mathbf{K}}_{u_r} = \widehat{\mathbf{K}}_r + \widehat{\Delta\mathbf{K}}_r \tag{3.49}$$

$$\widehat{\mathbf{K}}_r = \mathbf{T}^{\mathsf{T}}\widehat{\mathbf{K}}\mathbf{T} \tag{3.50}$$

$$\widehat{\Delta\mathbf{K}}_r = \mathbf{T}^{\mathsf{T}}\widehat{\Delta\mathbf{K}}\mathbf{T} \tag{3.51}$$

$$= \mathbf{T}^{\mathsf{T}}(\widehat{\mathbf{L}}\delta\mathbf{K}\widehat{\mathbf{L}}^{\mathsf{T}})\mathbf{T} \tag{3.52}$$

$$= \widehat{\mathbf{L}}_1\delta\mathbf{K}\widehat{\mathbf{L}}_1^{\mathsf{T}} \tag{3.53}$$

$$\widehat{\mathbf{L}}_1 = \mathbf{T}^{\mathsf{T}}\widehat{\mathbf{L}}. \tag{3.54}$$

The desired set of eigenvalues, $\mathbf{\Sigma}_1$, can be obtained by choosing the smallest N_o eigenvalues of (3.48) as

$$\mathbf{\Sigma}_1 = \widetilde{\mathbf{\Sigma}}(1 : N_o, 1 : N_o). \tag{3.55}$$

The corresponding eigenvectors can be obtained using (3.40) with

$$\mathbf{Q} = \widetilde{\mathbf{Q}}(1 : N_r, 1 : N_o). \tag{3.56}$$

(3.55) and (3.56) employ the so-called 'Matlab' notation according to which, $\mathbf{A}(a : b, c : d)$ refers to a sub-matrix formed by choosing rows a through b and columns c through d of a matrix $\mathbf{A}$.

The eigenvectors of the original uncertain system can then be obtained using (3.39). Algorithm 2 outlines the application of the proposed method to compute the statistics of the eigenproperties of the uncertain system along with the computational costs associated with each step. Thus, the total cost associated with the proposed method is

Algorithm 2 Proposed Method to Compute $\mathbf{\Psi}_1$ and $\mathbf{\Sigma}_1$

Require: $\mathbf{K}$, $\mathbf{M}$

1:	Solve the nominal eigenvalue problem, (3.34)	$(4n^3/3)$
2:	Compute $\mathbf{M}^{1/2}$	$(4n^3/3)$
3:	Compute $\mathbf{M}^{-1/2}$	(n)
4:	Compute $\widehat{\mathbf{K}}$ from (3.29)	$(2n^3)$
5:	Compute $\widehat{\mathbf{L}}$ from (3.33)	$(n^2 N_p)$
6:	Choose N_b	$(-)$
7:	Compute $\mathbf{T}$ from (3.47)	$(2nN_r^2)$
8:	Compute $\widehat{\mathbf{K}}_r$ from (3.50)	$(n^2 N_r + nN_r^2)$
9:	Compute $\widehat{\mathbf{L}}_1$ from (3.54)	$(nN_r N_p)$
10:	Generate samples of $\Delta\mathbf{K}$	$(-)$
11:	**for all** samples of $\Delta\mathbf{K}$ **do**	
12:	Compute $\widehat{\Delta\mathbf{K}}_r$ from (3.53)	$(N_p^2 N_r + N_p N_r^2)$
13:	Solve the reduced eigenvalue problem, (3.48)	$(4N_r^3/3)$
14:	Obtain $\mathbf{\Sigma}_1$ from (3.55)	$(-)$
15:	Obtain $\mathbf{Q}$ from (3.56)	$(-)$
16:	Compute $\widehat{\mathbf{\Psi}}_1$ from (3.40)	(nN_r^2)
17:	Compute $\mathbf{\Psi}_1$ from (3.39)	$(N_o n^2)$
18:	**end for**	
19:	Compute required statistics using (3.16)–(3.19)	

$$C_p = \frac{14}{3}n^3 + (N_p + N_r)n^2 + (3N_r^2 + N_pN_r + 1)n + N_s\left(\frac{4}{3}N_r^3 + (n + N_p)N_r^2 + N_p^2N_r + N_on^2\right). \tag{3.57}$$

It can be observed that when a large number of samples are to be computed, *i.e.*, N_s is large, the asymptotic computational cost of the proposed method is only $\mathcal{O}\left(N_sn^2\right)$ compared to $\mathcal{O}\left(N_sn_sn^2\right)$ of the brute-force method. Therefore, as will be demonstrated in Section 3.4, when the uncertainties are sufficiently localized ($N_p \ll n$) and a small number of modes of the system are desired ($N_o \ll n$), the proposed method yields considerable gains in computational efficiency.

3.3 Error Analysis of the Proposed Method

Since the eigenvalue problem considered herein is symmetric, the errors associated with the approximate eigenpair computed using Algorithm 2 can be predicted using an *a posteriori* error bound. Directly following the development in [202], let $\mathbf{A}$ be a symmetric matrix. Let $\widetilde{\lambda}$ and $\widetilde{\mathbf{u}}$ be an approximate eigenpair of the matrix $\mathbf{A}$, with $\|\widetilde{\mathbf{u}}\|_2 = 1$ and let $\mathbf{r}$ be the corresponding residual vector, given by

$$\mathbf{r} = \mathbf{A}\widetilde{\mathbf{u}} - \widetilde{\lambda}\widetilde{\mathbf{u}}. \tag{3.58}$$

Then, from the Bauer-Fike theorem (Theorem 3.6, Corollary 3.3 in [202]), there exists an eigenvalue, λ, of $\mathbf{A}$ such that

$$\left|\lambda - \widetilde{\lambda}\right| \leq \|\mathbf{r}\|_2 . \tag{3.59}$$

In addition, one can compute a bound on the angle, $\theta(\mathbf{u}, \widetilde{\mathbf{u}})$, between the true eigenvector, $\mathbf{u}$, and the corresponding approximation, $\widetilde{\mathbf{u}}$. It can be shown that (Theorem 3.9 in [202])

$$\sin\theta(\mathbf{u}, \widetilde{\mathbf{u}}) \leq \frac{\|\mathbf{r}\|_2}{\delta} \tag{3.60}$$

where $\delta = \min_i |\lambda_i - \widetilde{\lambda}|$, $i = 1, 2, \ldots, n$ and $\lambda_i \neq \lambda$; λ being the eigenvalue closest to $\widetilde{\lambda}$. This bound is not computable as the entire spectrum is generally not known. However, one can approximate δ by

$$\delta \approx \min_i \left| |\widetilde{\lambda}_i - \widetilde{\lambda}| - \|\mathbf{r}_i\|_2 \right| \tag{3.61}$$

where $i = 1, 2, \ldots, n$, $\widetilde{\lambda}_i$ is the approximation of the ith eigenvalue and $\mathbf{r}_i = \mathbf{A}\widetilde{\mathbf{u}}_i - \widetilde{\lambda}_i\widetilde{\mathbf{u}}_i$. The error bounds presented in this section can be utilized to choose the number of Krylov blocks, N_b, to be employed when applying Algorithm 2.

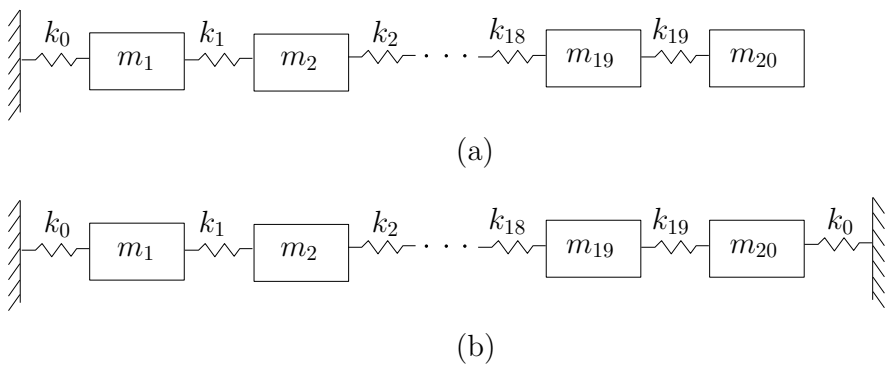
3.4 Numerical Examples

In this section, the numerical accuracy and the computational efficiency of the proposed method will be demonstrated using four examples. The first example considers Ram's [203] system. A parametric study is performed to investigate the effect of the number of Krylov blocks utilized to enrich the reduced basis, N_b, on the accuracy of Algorithm 2. The effect of the magnitude of the uncertainty in the system is also studied. Exact errors and error bounds on the eigenvalues and the eigenvectors computed using Algorithm 2 are utilized to demonstrate the numerical accuracy of the proposed method. The second example considers a cantilever beam with part of the beam resting on a Winkler foundation. A parametric study is performed to show the effect of N_b and the size of uncertainty on the accuracy of Algorithm 2. The effect of the size of uncertainty on the gain in computational efficiency is also studied. The third example considers a base-isolated 3D building. The Monte Carlo simulation method is used to estimate the modal characteristics of the building. The final example considers the computational model of a real structural system, the Cedar Avenue bridge, and demonstrates the computational efficiency of the proposed method. In addition to illustrating the use of the proposed method to compute modal characteristics, the final two examples also emphasize the gains in the computational efficiency of the proposed method for large-scale systems.

All numerical experiments were conducted using Matlab®, version 7.8.0.347 (R2009a). A Dell 490 Precision workstation, with a processor speed of 3.0 GHz and 8 GB of RAM was utilized for the first three examples while a machine equipped with two 2.40 GHz Intel Xeon E5530 quad-core CPUs with 16 GB of RAM was utilized for the final example. For all examples presented, $N_b = 2$.

3.4.1 Example 1: Ram's 20-DOF System

Consider a system comprising of 20 masses connected with springs [203], as shown in Figure 3.1. Figure 3.2 shows the relative errors in the first five eigenvalues of the



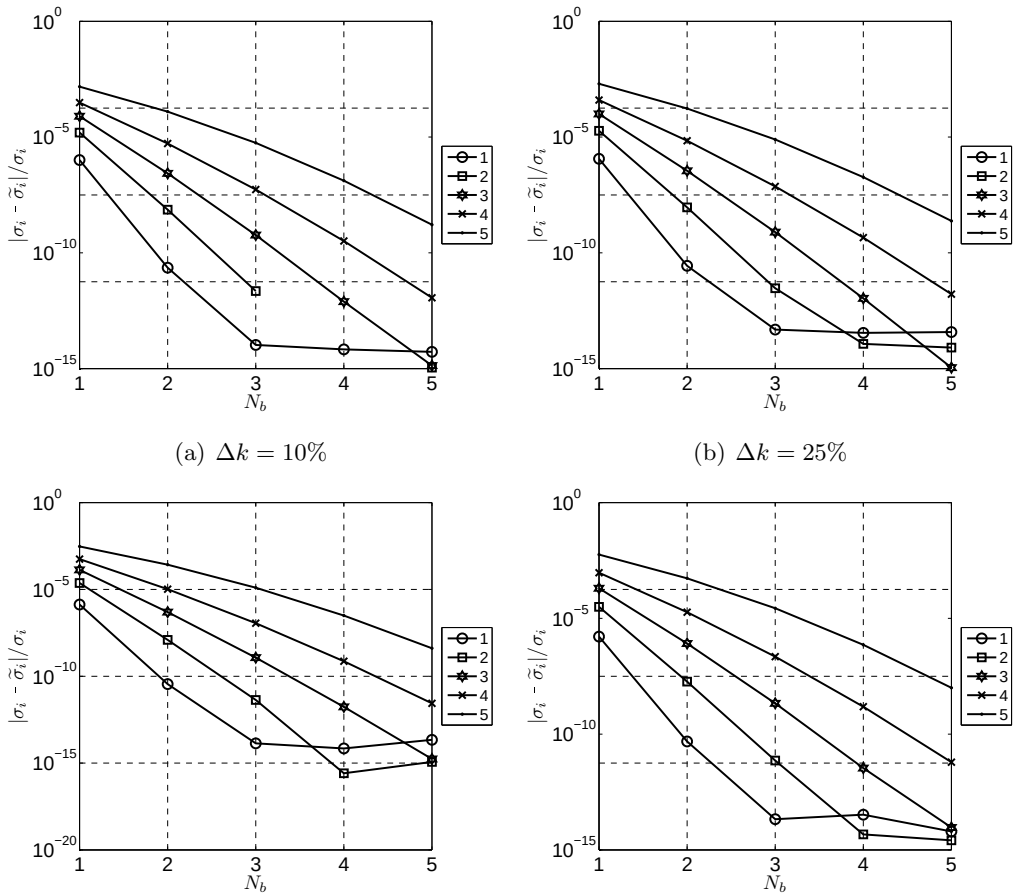
$$m_i = 1, i = 1, ..., 20; k_0 = 3000; k_i = 15000, i = 1, ..., 19$$

Figure 3.1: (a) Original System (b) Modified System

system as a function of the number of Krylov blocks, N_b. Figures 3.2(a)-3.2(d) show the behavior of these errors with respect to the magnitude of the perturbation in the system. ΔK represents the percent change in the stiffness of the spring, k_0, attached to the mass m_{20}. Figure 3.3 shows corresponding behavior of the errors in the computation of the eigenvectors. The errors in the eigenvalues and the eigenvectors decrease as N_b increases, which is expected. Moreover, the errors in higher modes decay slower than those in lower modes. Further, it can be observed that the effect of the magnitude of the perturbation is almost negligible on the numerical accuracy of Algorithm 2. Tables 3.1-3.2 show the error bounds and exact errors for the eigenvalues and the eigenvectors computed using Algorithm 2 for 5 modes and 10 modes being computed respectively.

3.4.2 Example 2: Cantilever Beam Model

A cantilever beam partially supported on a Winkler foundation is considered, as shown in Figure 3.4. The mass and the stiffness matrices for the beam are generated using a finite element formulation using standard 2-dimensional Eüler-Bernoulli beam elements. The beam is discretized with 101 nodes, which corresponds to $n = 200$ active



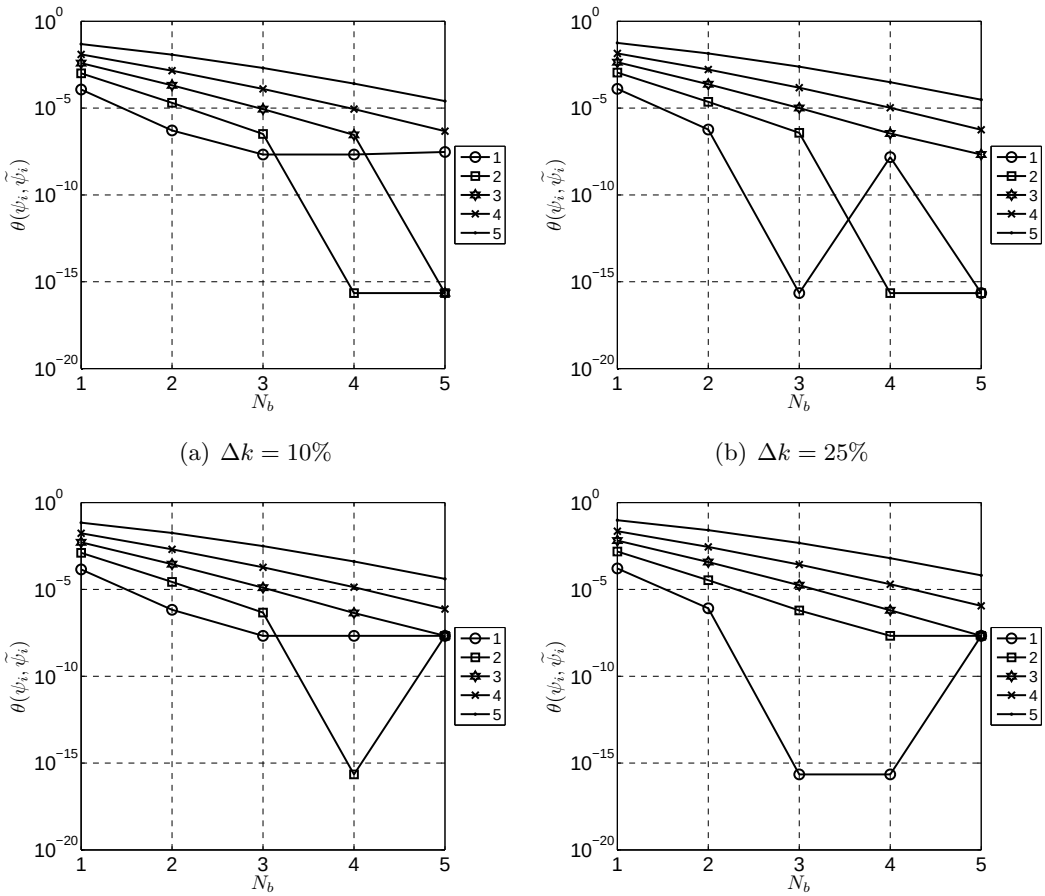
(a) $\Delta k = 10\%$ (b) $\Delta k = 25\%$

(c) $\Delta k = 50\%$ (d) $\Delta k = 100\%$

Figure 3.2: Effect of N_b and Δk on the Relative Error in Eigenvalues (Example 1)

Table 3.1: Error Bounds and Exact Errors for Example 1 (5 modes)

		Eigenvalue Errors		**Eigenvector Errors**	
i	$\widetilde{\sigma}_i$	$\frac{\lvert\sigma_i - \widetilde{\sigma}_i\rvert}{\sigma_i}$	$\lVert \mathbf{r}_i \rVert_2$	$\theta(\boldsymbol{\psi}_i, \widetilde{\boldsymbol{\psi}}_i)$	$\frac{\lVert \mathbf{r}_i \rVert_2}{\delta}$
1	1.9630E+02	2.8106E-11	1.9803E-07	5.8755E-07	1.7422E-05
2	8.4905E+02	9.3803E-09	2.7426E-04	2.3100E-05	6.4813E-04
3	2.0680E+03	3.5841E-07	1.2527E-02	2.3893E-04	3.2059E-03
4	3.9100E+03	7.1444E-06	2.6627E-01	1.6787E-03	1.2034E-02
5	6.3719E+03	1.7648E-04	1.3986E+01	1.4314E-02	1.1757E-01



(a) $\Delta k = 10\%$ (b) $\Delta k = 25\%$

(c) $\Delta k = 50\%$ (d) $\Delta k = 100\%$

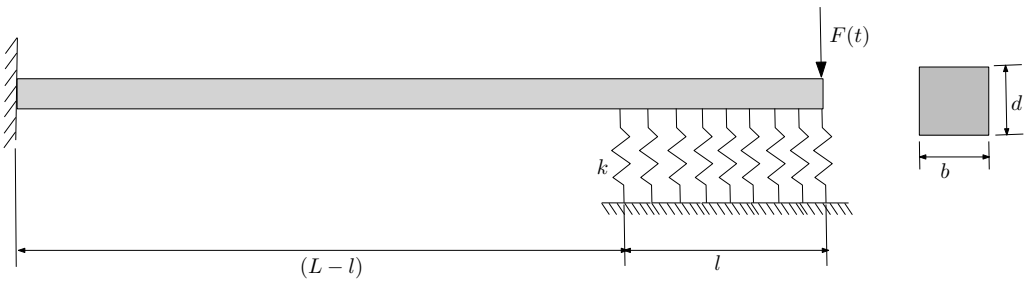
Figure 3.3: Effect of N_b and Δk on the Error in Eigenvectors (Example 1)

degrees-of-freedom. The nominal system for this example is simply the cantilever beam without any foundation.

The stiffness of the individual springs used to model the Winkler foundation is assumed to be uncertain, with a uniform distribution between $3EI/L^3$ and $12EI/L^3$. In terms of prescribing the $\mathbf{\Delta K}$ matrix, the uncertain Winkler foundation translates to adding nodal springs of uncertain stiffnesses to the beam which, in turn, corresponds to adding uncertain terms to the elements of the main diagonal of the system's stiffness matrix. This example was used to investigate the computational efficiency of the

Table 3.2: Error Bounds and Exact Errors for Example 1 (10 modes)

		Eigenvalue Errors		**Eigenvector Errors**	
i	$\widetilde{\sigma}_i$	$\frac{\lvert\sigma_i-\widetilde{\sigma}_i\rvert}{\sigma_i}$	$\lVert\mathbf{r}_i\rVert_2$	$\theta(\boldsymbol{\psi}_i, \widetilde{\boldsymbol{\psi}}_i)$	$\frac{\lVert\mathbf{r}_i\rVert_2}{\delta}$
1	2.0261E+02	2.8694E-14	5.9505E-11	0.0000E+00	2.9851E-07
2	8.7021E+02	1.3782E-12	7.4955E-08	1.7501E-07	1.0594E-05
3	2.1034E+03	3.9714E-11	2.7286E-06	1.4869E-06	4.7027E-05
4	3.9546E+03	4.4043E-10	3.6037E-05	6.9846E-06	1.3949E-04
5	6.4196E+03	3.0519E-09	2.8330E-04	2.4406E-05	3.3892E-04
6	9.4570E+03	1.6732E-08	1.6844E-03	7.3414E-05	7.4456E-04
7	1.3001E+04	8.4046E-08	8.7539E-03	2.0857E-04	1.5717E-03
8	1.6967E+04	4.3804E-07	4.4646E-02	6.0735E-04	3.3561E-03
9	2.1262E+04	2.7912E-06	2.5670E-01	2.0324E-03	7.7406E-03
10	2.5781E+04	3.1913E-05	3.6315E+00	1.0467E-02	3.6364E-02



$E = 30 \times 10^6 \text{psi}, b = d = 3\text{in}, L = 30\text{ft}, \rho = 0.294\text{pci}, \quad k = Uniform(3, 12) \times \frac{EI}{L^3}$

Figure 3.4: Example Cantilever Beam

proposed method. Various cases were considered to test the efficiency of the proposed method under different amounts of uncertainty present in the system. The effect of the number of degrees-of-freedom whose output is desired on the efficiency of the method was also studied. To vary the amount of uncertainty present in the system, the length of the beam supported on the Winkler foundation, l, was increased from 1% to 10% of the length of the beam. The number of output degrees-of-freedom was varied by selecting equidistant points on the beam whose output was computed. In terms of the nomenclature used in the expected efficiency gain functions, this corresponds to N_p being varied from 2% to 20% of n.

Tables 3.3-3.5 show the gains in computational efficiency obtained using the proposed method for the computation of the statistics of eigenvalues and eigenvectors of the cantilever beam, using 100, 1000 and 10000 samples respectively. It can be seen that there is an $\mathcal{O}(10)$ gain for 10% of the system being uncertain, and $\mathcal{O}\left(10^2\right)$ gains can be obtained when the system has smaller amount of uncertainty. It should be noted that this is a moderate sized system, and the computational gains are expected to increase with larger systems. This behavior will be demonstrated in the subsequent numerical examples.

Table 3.3: Gains in Computational Efficiency for Example 2 (100 samples)

	I		**II**	
# of modes computed $\Rightarrow$	10	20	10	20
$l = 1\%$ of n	20.17	32.86	17.57	31.57
$l = 5\%$ of n	17.57	37.83	11.73	24.89
$l = 10\%$ of n	9.38	8.76	9.46	8.58

I: computation of statistics of eigenvalues only
II: computation of statistics of both eigenvalues and eigenvectors

Table 3.4: Gains in Computational Efficiency for Example 2 (1,000 samples)

	I		**II**	
# of modes computed $\Rightarrow$	10	20	10	20
$l = 1\%$ of n	110.64	70.45	103.08	81.78
$l = 5\%$ of n	39.23	36.83	29.61	34.67
$l = 10\%$ of n	16.54	11.57	15.63	10.68

I: computation of statistics of eigenvalues only
II: computation of statistics of both eigenvalues and eigenvectors

Figure 3.5 shows the behavior of the errors in the first five eigenvalues as a function of the number of Krylov blocks, N_b, and the size of the uncertainty. It can be observed that there is no significant effect of the size of uncertainty on the numerical accuracy of Algorithm 2.

Table 3.5: Gains in Computational Efficiency for Example 2 (10,000 samples)

	I		**II**	
# of modes computed ⇒	10	20	10	20
$l = 1\%$ of n	112.07	83.84	75.15	70.81
$l = 5\%$ of n	39.87	38.21	35.57	35.85
$l = 10\%$ of n	17.55	10.90	15.23	10.61

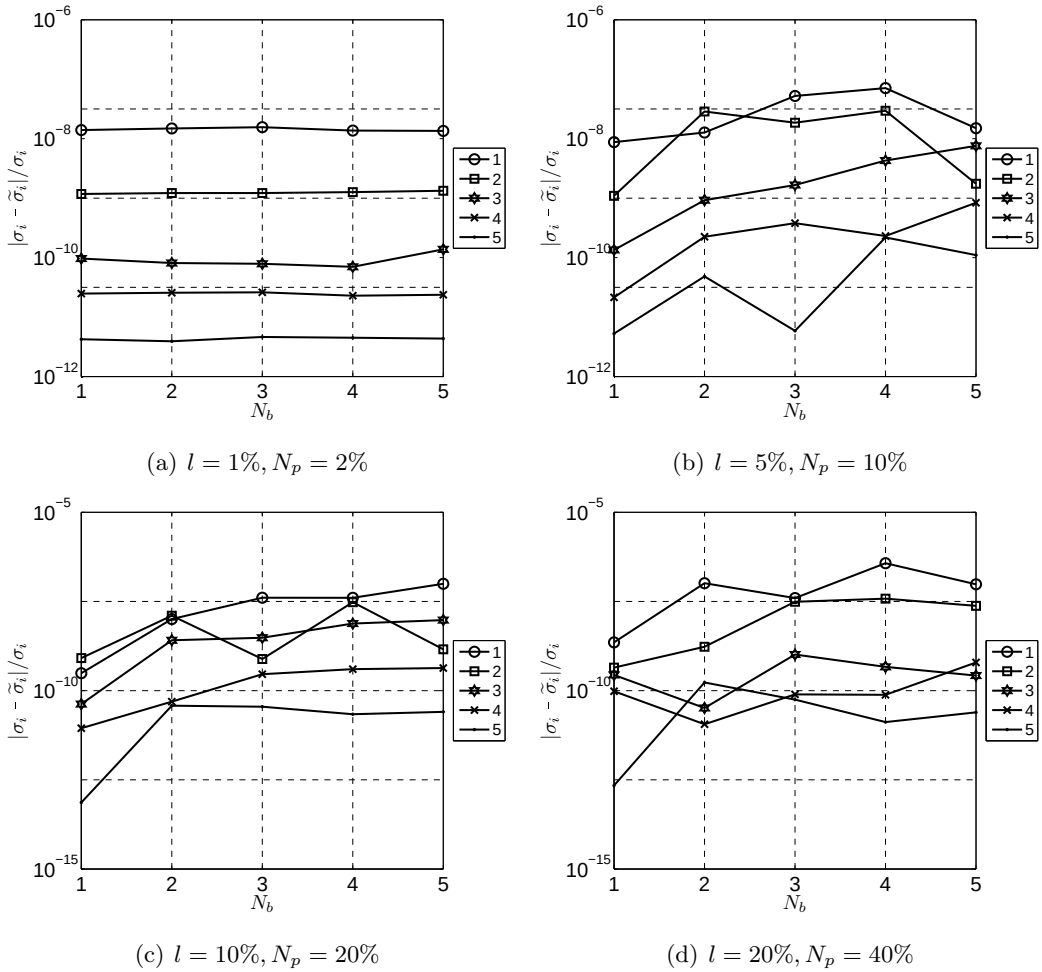
I: computation of statistics of eigenvalues only
II: computation of statistics of both eigenvalues and eigenvectors

3.4.3 Example 3: 3D Building Model with Base Isolation Layer

A 615 degree-of-freedom three-dimensional steel building, as shown in Figure 3.6, is considered to illustrate the computation of stochastic response characteristics using the proposed method. The building is modeled using a finite element formulation consisting of beam and truss elements. The building is assumed to be isolated from ground through a bed of translational and rotational springs and a rigid block mass, M_b. This isolation bed is subsequently lumped into three springs and dashpots at the center of mass of the base block. The FE model of the building itself consists of 612 degrees-of-freedom while the remaining 3 degrees-of-freedom are associated with the rigid mass (two translations and one rotation). The stiffnesses of the translational and the rotational springs that connect the rigid mass to the ground are assumed to be independent random variables as described in Figure 3.6.

Table 3.6 shows the required computational times and gains in the computational efficiency for $10,000$ sample realizations of the first few eigenpairs of the building model. It can be seen that the proposed method takes significantly lesser computational time compared to the brute-force method. The proposed method achieves an $\mathcal{O}\left(10^2\right)$ gain in computational efficiency for all the four cases considered.

Figure 3.7 shows the histograms of the first four eigenvalues of the system along with the locations of the respective means. Once the histograms have been computed, expert judgement along with trial and error can be used to postulate an appropriate probability density function for the variables of interest.



(a) $l = 1\%, N_p = 2\%$

(b) $l = 5\%, N_p = 10\%$

(c) $l = 10\%, N_p = 20\%$

(d) $l = 20\%, N_p = 40\%$

Figure 3.5: Effect of N_b and N_p on the Error in Eigenvalues (Example 2)

3.4.4 Example 4: Cedar Avenue Bridge Model

The final example considers the computational model of an actual bridge, the Cedar Avenue bridge, in order to demonstrate the computational efficiency of the proposed method when applied to a real life system. The bridge is located in the metro area of the Twin Cities in Minnesota. Thompson and Schultz [204] created a finite element model of the Cedar Avenue Bridge in SAP2000 [205] using 3D frame and shell elements. Figure 3.8 shows a photograph and the finite element model of the Cedar Avenue bridge.

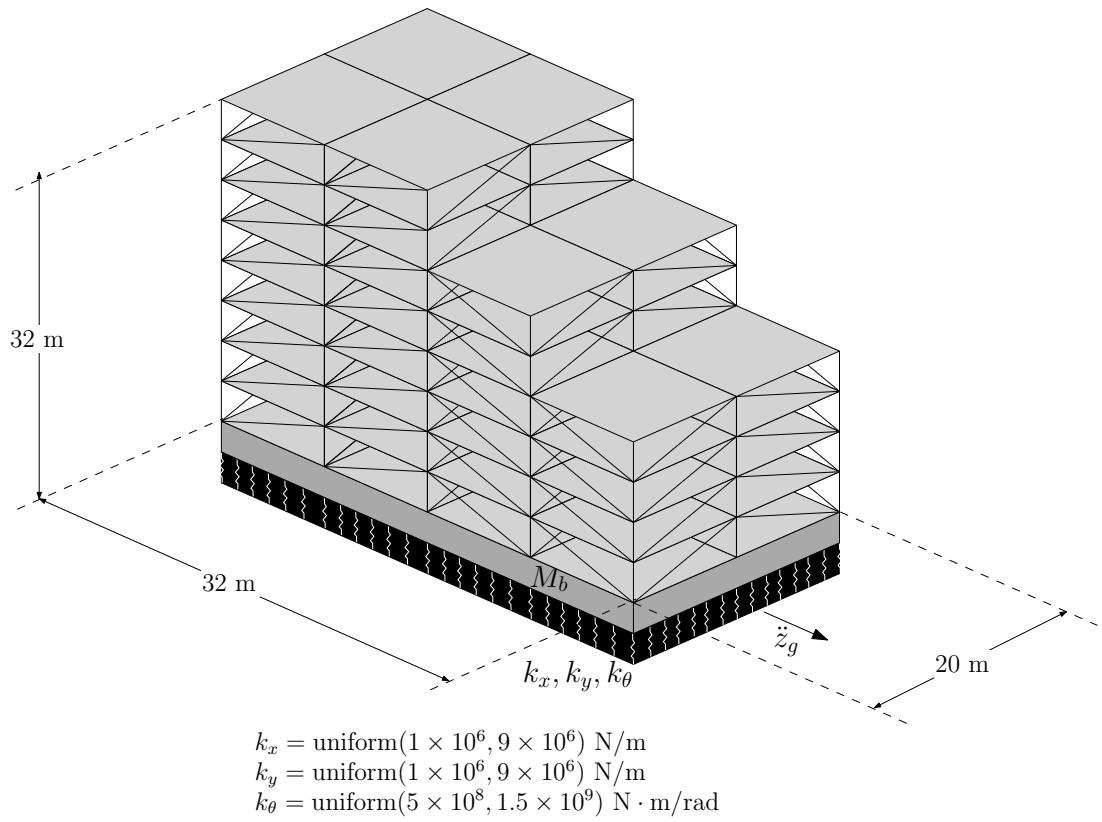


Figure 3.6: 3D Building with Base Isolator Layer

The bridge model is equipped with a scissor jack apparatus as shown in Figure 3.9(a) to provide counter moments in order to reduce the moment range experienced by certain joints. Gastineau *et al.* [206] developed the finite element model of the modified bridge consisting of $17,687$ degrees-of-freedom as shown in Figure 3.9(b). The modification apparatus is connected across four joints, thereby causing a rank-8 perturbation in the system's stiffness matrix. The stiffness properties of the modification apparatus are assumed to be uncertain, modeled using a uniform distribution within a 4% range of the corresponding nominal values.

Table 3.7 shows the required computational times and gains in computational efficiency for $1,000$ and $10,000$ sample realizations of the computation of eigenvalues and corresponding eigenvectors of the system. The proposed method achieves $\mathcal{O}\left(10^3\right)$

Table 3.6: Computational Times and Efficiency Gain Ratios for Example 3 (for $10,000$ samples)

	I		**II**	
# of modes computed⇒	10	20	10	20
Brute-force method CPU time (t_b)	3337.84 sec (55.63 min)	4374.47 sec (72.91 min)	3335.77 sec (55.60 min)	4406.36 sec (73.44 min)
Proposed method CPU time (t_p)	4.41 sec	8.38 sec	5.94 sec	11.55 sec
Computational efficiency (t_b/t_p)	757.52	522.32	561.81	381.61

I: computation of statistics of eigenvalues only
II: computation of statistics of both eigenvalues and eigenvectors

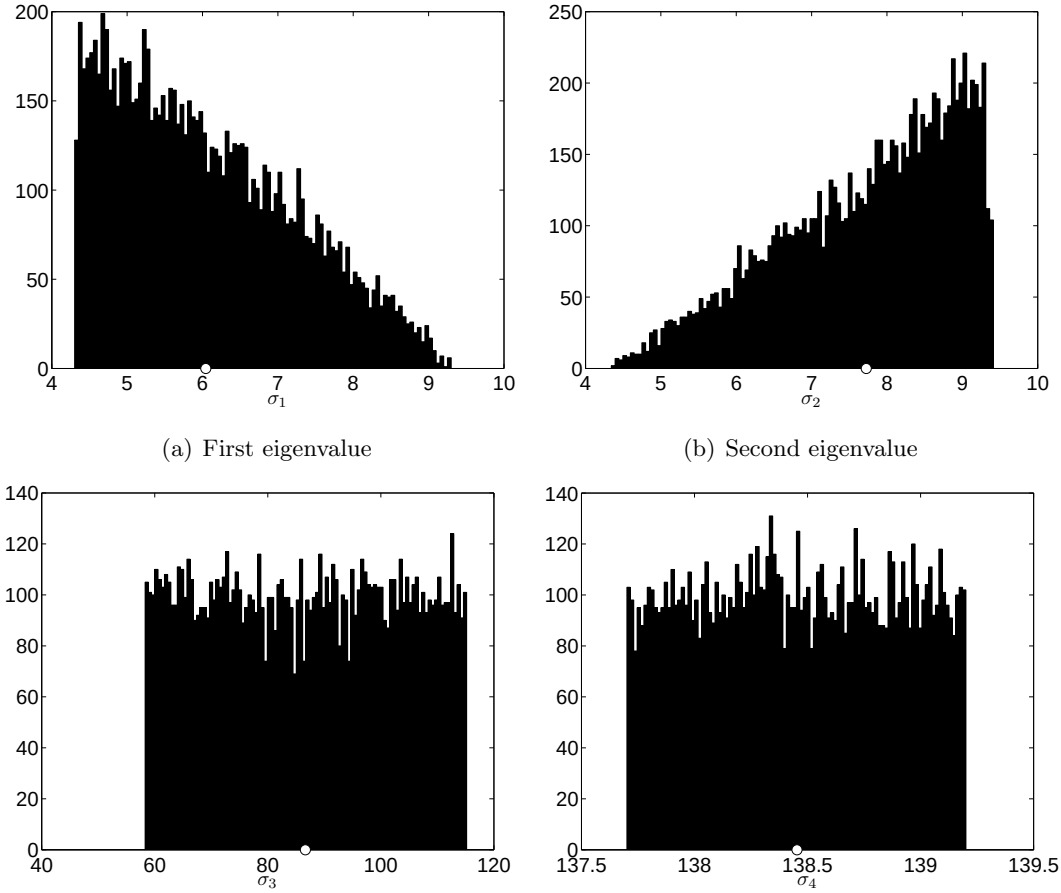
gain in computational efficiency when $10,000$ sample realizations are computed.

Table 3.7: Computational Times and Efficiency Gain Ratios for Example 4

	1,000 samples		**10,000 samples**	
# of modes computed⇒	2	10	2	10
Brute-force method CPU time (t_b)	4.6 days	5 days	1.5 months	1.7 months
Proposed method CPU time (t_p)	14 minutes	15 minutes	19 minutes	22 minutes
Computational efficiency (t_b/t_p)	488	488	3410	3416

3.5 Conclusions

This chapter presented the development of an efficient computational methodology for the modal analysis of linear systems with local stiffness uncertainties. The proposed method, outlined in Algorithm 2, assumes that the sub-spectrum of interest of a nominal system has already been computed. The relevant sub-spectrum of the related systems was computed by projecting the associated eigenvalue problems in a reduced dimensional linear generalized coordinate space. The basis of this reduced dimensional space was constructed by augmenting the sub-spectrum of the nominal system with additional



(a) First eigenvalue

(b) Second eigenvalue

(c) Third eigenvalue

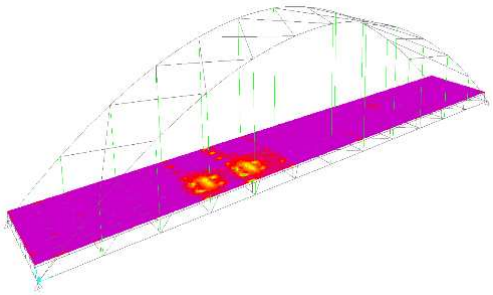
(d) Fourth eigenvalue

Figure 3.7: Histograms of First Four Eigenvalues (Example 3)

enrichment vectors that utilized the knowledge about the pattern of the modifications of the related systems. Theoretical error bounds on the eigenvalues and the eigenvectors of the related systems computed using the proposed method were also discussed along with the theoretical computational costs associated with the proposed method.

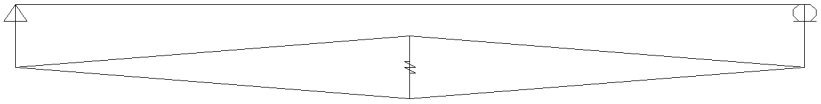
Four numerical examples were presented to highlight different aspects of the proposed method. The first example demonstrated the numerical accuracy of the proposed method along with the usefulness of the error bounds in predicting the expected errors in the approximations. In addition, a parametric study was also

(a) Photograph of the Bridge

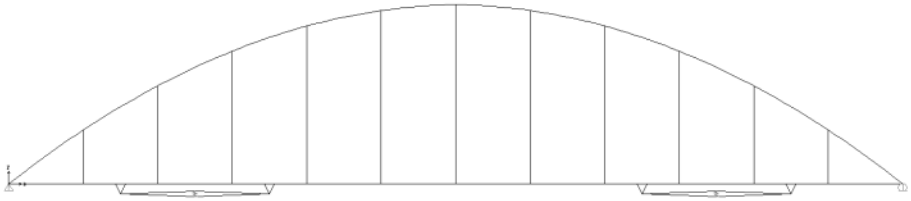


(b) FE Model of the Bridge [204]

Figure 3.8: Example 4: Cedar Avenue Bridge Model



(a) Scissor-Jack Apparatus [206]



(b) Model of the Bridge with Apparatus [206]

Figure 3.9: Modified Bridge Model

performed to investigate the effect of the size of the reduced dimensional linear space along with the magnitude of the modification of the system on the associated errors in the computation of the eigenvalues and the eigenvectors.

The second example demonstrated the effect of the size of uncertainty on both the computational efficiency and the numerical accuracy of the proposed method. The

behavior of the errors in the approximate eigenvalues and the eigenvectors with respect to the size of the reduced dimensional space was also shown. The parametric studies performed for the first two examples showed that the numerical accuracy of the proposed method was not affected by either the magnitude or the size of the uncertainty in the system. In real applications, such parametric studies can be utilized to estimate the size of the reduced dimensional basis that would be required in order to satisfy a given error tolerance.

The third example showed the application of the proposed method to a moderate-sized computational model with local stiffness uncertainties. A gain in computational efficiency of $\mathcal{O}\left(10^2\right)$ was observed. Histograms of the first four eigenvalues were also presented which could be utilized to postulate relevant probabilistic description of the outputs of interest. The final example considered the computational model of a real structure; a gain in computational efficiency of $\mathcal{O}\left(10^3\right)$ was observed. As hoped, the gain in computational efficiency demonstrated by the proposed method was magnified for the larger dimensional computational model.

Chapter 4

Frequency Domain Response of Systems

This chapter discusses the development of an exact method for the frequency domain analysis of an LTI system with local uncertainties in damping and stiffness characteristics. The proposed method is not restricted by the structure of the system uncertainties as long as the system remains stable with probability one. In general, the system matrices can be non-symmetric and the system is not required to have classical damping. Moreover, there is no effect of the magnitude of the uncertainties on the accuracy of the solution. The proposed method can be utilized to efficiently compute the relevant statistics related to the transfer function matrix of a system and/or the PSD of the system's response when subjected to a stationary random input given as its PSD.

The equations of motion of an n-dimensional LTI system can be written in the frequency domain as

$$\left(\mathbf{M}s^2 + \mathbf{D}s + \mathbf{K}\right)\mathbf{X}(s) = \mathbf{F}(s) \tag{4.1}$$

where $\mathbf{M}, \mathbf{D}$, and $\mathbf{K} \in \mathbb{R}^{n \times n}$ are the mass, the damping, and the stiffness matrices of the system, respectively; $\mathbf{X}(s)$ is the response of the system in the frequency domain; $\mathbf{F}(s)$ is the input to the system in the frequency domain; and s is a complex frequency domain parameter. The system given by (4.1) will be referred to as the 'nominal' system in the remainder of this chapter. It should be noted that unlike the previous chapter,

no assumptions regarding the symmetry or the positive-definiteness of the mass, the stiffness and the damping matrices is necessary for the developments in this chapter. The frequency domain response of (4.1) can be written as

$$\mathbf{X}(s) = \mathbf{H}(s)\mathbf{F}(s) \tag{4.2}$$

where $\mathbf{H}(s) \in \mathbb{C}^{n\times n}$ is the system's transfer function matrix given by

$$\mathbf{H}(s) = \left(\mathbf{M}s^2 + \mathbf{D}s + \mathbf{K}\right)^{-1}. \tag{4.3}$$

When the input to the system, $\mathbf{F}(s)$, is random and prescribed by its PSD, $\mathbf{S_{ff}}(\omega)$, the PSD of the response can be directly computed as

$$\mathbf{S_{xx}}(\omega) = \mathbf{H}(\omega)\mathbf{S_{ff}}(\omega)\mathbf{H}^{\mathsf{H}}(\omega) \tag{4.4}$$

where $\mathbf{H}(\omega) \in \mathbb{C}^{n\times n}$ is obtained by substituting $s = \imath\omega$ in (4.3).

The equations of motion of an LTI system with stiffness and damping uncertainties can be represented in a form similar to (4.1) as

$$\left[\mathbf{M}s^2 + \mathbf{D}_u s + \mathbf{K}_u\right]\mathbf{V}(s) = \mathbf{F}(s) \tag{4.5}$$

where the mass matrix, $\mathbf{M}$, is the same as that in (4.1) and the random stiffness and damping matrices, $\mathbf{D}_u$ and $\mathbf{K}_u$, can be represented as perturbed forms of the nominal stiffness and damping matrices, $\mathbf{D}$ and $\mathbf{K}$, as

$$\mathbf{D}_u = \mathbf{D} + \Delta\mathbf{D} \tag{4.6}$$

$$\mathbf{K}_u = \mathbf{K} + \Delta\mathbf{K} \tag{4.7}$$

where $\Delta\mathbf{D}, \Delta\mathbf{K} \in \mathbb{R}^{n\times n}$ are matrices of ranks N_{p_d} and N_{p_k}, respectively, and are assumed to contain all of the uncertainty of the system. Since the uncertainties have been assumed to be local in nature, $N_{p_d} \ll n$ and $N_{p_k} \ll n$ which implies that $\Delta\mathbf{D}$ and $\Delta\mathbf{K}$ are relatively sparse matrices and can be represented without loss of generality as

$$\Delta\mathbf{D} = \mathbf{L}_d\delta\mathbf{D} \tag{4.8}$$

$$\Delta\mathbf{K} = \mathbf{L}_k\delta\mathbf{K} \tag{4.9}$$

where $\delta\mathbf{D} \in \mathbb{R}^{N_{p_d}\times n}$ and $\delta\mathbf{K} \in \mathbb{R}^{N_{p_k}\times n}$ are dense matrices containing the uncertainties in the damping and the stiffness of the system, respectively. $\mathbf{L}_d \in \mathbb{R}^{n\times N_{p_d}}$ and $\mathbf{L}_k \in \mathbb{R}^{n\times N_{p_k}}$

denote the positions of these uncertainties in the system and are permutation matrices comprising of zeros and ones. It should be noted that the representation of $\Delta\mathbf{D}$ and $\Delta\mathbf{K}$ as in (4.8) and (4.9), respectively, allows for non-symmetric perturbations in the system unlike the representation used for $\Delta\mathbf{K}$ in the previous chapter. Such perturbations may take the form of modifying entire rows in the $\mathbf{D}$ and the $\mathbf{K}$ matrices instead of just modifying a diagonal block, as in the situation when a feedback control is imposed on the system.

It is possible to combine the matrices $\mathbf{L}_d$ and $\mathbf{L}_k$ in one matrix, $\mathbf{L}$, as

$$\mathbf{L} = \mathbf{L}_d \cup \mathbf{L}_k \tag{4.10}$$

where '$\cup$' signifies the combination of only the unique columns of the matrices $\mathbf{L}_d$ and $\mathbf{L}_k$. $\mathbf{L}$ is of size $n \times N_p$ where N_p is the total number of unique columns in $\mathbf{L}_d$ and $\mathbf{L}_k$. $\delta\mathbf{D}$ and $\delta\mathbf{K}$ are transformed to matrices of size $N_p \times n$ containing zero rows at appropriate locations. In the worst case, when the uncertainties in the damping and that in the stiffness are present at completely different locations, these combined matrices can be formed with a slight abuse of notation as

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_d & \mathbf{L}_k \end{bmatrix}, \quad \delta\mathbf{D} = \begin{bmatrix} \delta\mathbf{D} \\ \mathbf{0}_{N_{p_k} \times n} \end{bmatrix}, \quad \delta\mathbf{K} = \begin{bmatrix} \mathbf{0}_{N_{p_d} \times n} \\ \delta\mathbf{K} \end{bmatrix}. \tag{4.11}$$

In the simplest case, when the locations of the uncertainties in the damping and the stiffness coincide, *i.e.*, $\mathbf{L}_d = \mathbf{L}_k$, the structure of the matrices $\delta\mathbf{D}$ and $\delta\mathbf{K}$ remains unchanged and

$$\mathbf{L} = \mathbf{L}_d = \mathbf{L}_k. \tag{4.12}$$

Using the combined $\mathbf{L}$ matrix, the uncertain portions of the damping and the stiffness matrices, $\Delta\mathbf{D}$ and $\Delta\mathbf{K}$, can be rewritten as

$$\Delta\mathbf{D} = \mathbf{L}\delta\mathbf{D} \tag{4.13}$$

$$\Delta\mathbf{K} = \mathbf{L}\delta\mathbf{K}. \tag{4.14}$$

In the remainder of the chapter, (4.13) and (4.14) will be utilized to represent the system uncertainties instead of (4.8) and (4.9) for the sake of brevity. Although the size and the structure of $\delta\mathbf{D}$ and $\delta\mathbf{K}$ matrices are different in these two sets of equations, the same mathematical symbols have been employed to represent these matrices. In general, $\delta\mathbf{D}$

and $\delta\mathbf{K}$ will represent $N_p \times n$ matrices, as in (4.13) and (4.14) unless otherwise noted; a distinction in notation will be made where necessary.

The following section will discuss a conventional method to compute the statistics of the relevant components of the transfer function matrix and the PSD of the system's response. It will be referred to as the 'brute-force method' in the remainder of the chapter. Next, a new approach will be proposed for the frequency domain analysis of LTI systems with local uncertainties in damping and stiffness. A specialized case of the proposed method, applicable to scenarios where the uncertainties arise only as diagonal blocks, will also be discussed. Finally, numerical examples will be presented to demonstrate the computational efficiency of the proposed method as compared to that of the brute-force method.

4.1 Brute-Force Method

The frequency domain response, the transfer function matrix and the PSD of the response of the uncertain system can be computed, respectively, as

$$\mathbf{V}(s) = \mathbf{H}_u(s)\mathbf{F}(s) \tag{4.15}$$

$$\mathbf{H}_u(s) = \left(\mathbf{M}s^2 + \mathbf{D}_u s + \mathbf{K}_u\right)^{-1} \tag{4.16}$$

$$\mathbf{S}_{\mathbf{vv}}(\omega) = \mathbf{H}_u(\omega)\mathbf{S}_{\mathbf{ff}}(\omega)\mathbf{H}_u^{\mathsf{H}}(\omega). \tag{4.17}$$

When the input to the system is present only at $N_f < n$ degrees-of-freedom and the output of only $N_o < n$ degrees-of-freedom of the system is required, (4.17) can be rewritten as

$$\mathbf{S}_{\mathbf{vv}} = \mathbf{H}_u(1:N_o, 1:N_f)\mathbf{S}_{\mathbf{ff}}\mathbf{H}_u^{\mathsf{H}}(1:N_o, 1:N_f). \tag{4.18}$$

(4.18) again utilizes the Matlab-notation introduced in Chapter 3 and is computationally more efficient compared to (4.17) when the response of the entire system is not required. In this special case, it should be noted that the size of $\mathbf{S}_{\mathbf{ff}}$ and $\mathbf{S}_{\mathbf{vv}}$ will be $N_f \times N_f$ and $N_o \times N_o$, respectively. Employing a sampling-based approach, the mean of the transfer function matrix and the PSD of the system's response can be computed using the

following unbiased estimators:

$$\boldsymbol{\mu}_{\mathbf{H}_u}(s) = \frac{1}{N_s}\sum_{j=1}^{N_s}\mathbf{H}_u^{(j)}(s) \tag{4.19}$$

$$\boldsymbol{\mu}_{\mathbf{S_{vv}}}(\omega) = \frac{1}{N_s}\sum_{j=1}^{N_s}\mathbf{S}_{\mathbf{vv}}^{(j)}(\omega) \tag{4.20}$$

where N_s is the number of samples computed. Algorithm 3 outlines the brute-force procedure to compute the statistics of the transfer function matrix and PSD of the system's response. The computational cost associated with a particular step is listed in parentheses. The total computational cost associated with computing all the samples of the transfer function matrix and PSD of the system's response will respectively be

$$C_{b,H} = N_{freq}\left[\frac{8}{3}N_s n^3\right] \tag{4.21}$$

$$C_{b,S} = N_{freq}\left[N_s\left(\frac{8}{3}n^3 + 4N_o^2 N_f + 4N_o N_f^2\right)\right]. \tag{4.22}$$

Algorithm 3 Brute-Force Method to Compute $\mathbf{V}(s)$ and/or $\mathbf{S_{VV}}(\omega)$

Require: $\mathbf{M}$, $\mathbf{D}$, $\mathbf{K}$

```
Populate complex parameter s in desired range
Generate samples of ΔD and ΔK                                  (−)
for all samples of ΔD and ΔK do
    for all s do
        Compute D_u and K_u using (4.6) and (4.7)              (−)
        Compute H_u using (4.16)                               (8n^3/3)
        Compute S_vv using (4.18)                              (4N_o^2 N_f + 4N_o N_f^2)
    end for
end for
Compute statistics from the generated samples
```

4.2 Proposed Method

The equations of motion of the uncertain system given by (4.5) can be rewritten using (4.6) and (4.7) as

$$\left[\left(\mathbf{M}s^2 + \mathbf{D}s + \mathbf{K}\right) + (\Delta\mathbf{D}s + \Delta\mathbf{K})\right]\mathbf{V}(s) = \mathbf{F}(s). \tag{4.23}$$

(4.23) can again be rewritten exploiting the form of the uncertain matrices, $\Delta\mathbf{D}$ and $\Delta\mathbf{K}$, given by (4.13) and (4.14) as

$$\left[\left(\mathbf{M}s^2 + \mathbf{D}s + \mathbf{K}\right) + \mathbf{L}(\delta\mathbf{D}s + \delta\mathbf{K})\right]\mathbf{V}(s) = \mathbf{F}(s). \tag{4.24}$$

Let the frequency domain response of the uncertain system, $\mathbf{V}$, be represented in terms of the response of the nominal system as

$$\mathbf{V}(s) = \mathbf{X}(s) + \widetilde{\mathbf{H}}(s)\mathbf{P}(s) \tag{4.25}$$

where

$$\widetilde{\mathbf{H}}(s) = \mathbf{H}(s)\mathbf{L} \tag{4.26}$$

and $\mathbf{P}$ is an unknown force accounting for the uncertainties in the damping and the stiffness. It should be noted that the computation of $\widetilde{\mathbf{H}}$ in (4.26) will not require a matrix-matrix multiplication; $\widetilde{\mathbf{H}}$ can be obtained by choosing appropriate columns of the matrix $\mathbf{H}$. From (4.1), (4.24) and (4.25), $\mathbf{P}$ is given by

$$\mathbf{P} = -[\mathbf{I}_{N_p\times N_p} + (\delta\mathbf{D}s + \delta\mathbf{K})\widetilde{\mathbf{H}}]^{-1}(\delta\mathbf{D}s + \delta\mathbf{K})\mathbf{X}. \tag{4.27}$$

From (4.25) and (4.27), the frequency domain response of the uncertain system is given by

$$\mathbf{V} = \left[\mathbf{I}_{n\times n} - \widetilde{\mathbf{H}}[\mathbf{I}_{N_p\times N_p} + (\delta\mathbf{D}s + \delta\mathbf{K})\widetilde{\mathbf{H}}]^{-1}(\delta\mathbf{D}s + \delta\mathbf{K})\right]\mathbf{HF}. \tag{4.28}$$

From (4.28), the transfer function matrix, $\mathbf{H}_u$, of the uncertain system is simply

$$\mathbf{H}_u = \left[\mathbf{I}_{n\times n} - \widetilde{\mathbf{H}}[\mathbf{I}_{N_p\times N_p} + (\delta\mathbf{D}s + \delta\mathbf{K})\widetilde{\mathbf{H}}]^{-1}(\delta\mathbf{D}s + \delta\mathbf{K})\right]\mathbf{H}. \tag{4.29}$$

Once the transfer function matrix has been computed, the PSD of the response of the system can be directly computed from (4.17). Relevant statistics can then be computed using (4.19) and (4.20). It should be noted that (4.29) can be utilized to compute

Algorithm 4 Proposed Method to Compute $\mathbf{V}(s)$ and/or $\mathbf{S_{VV}}(\omega)$

Require: $\mathbf{M}$, $\mathbf{D}$, $\mathbf{K}$

Populate complex parameter s in desired range
Compute $\mathbf{H}$ using (4.3) $((8/3)N_{freq}n^3)$
Generate samples of $\Delta\mathbf{D}$ and $\Delta\mathbf{K}$ $(-)$
for all samples of $\Delta\mathbf{D}$ and $\Delta\mathbf{K}$ **do**
  for all s **do**
    Compute $\delta\mathbf{D}$ and $\delta\mathbf{K}$ conforming to (4.13) and (4.14) $(-)$
    Compute $\widetilde{\mathbf{H}}(s)$ using (4.26) $(-)$
    Compute auxiliary matrix $\mathbf{B}_1 = (\delta\mathbf{D}s + \delta\mathbf{K})\widetilde{\mathbf{H}}$ $(4nN_p^2)$
    Compute auxiliary matrix $\mathbf{B}_2 = [\mathbf{I}_{N_p \times N_p} + \mathbf{B}_1]^{-1}$ $(8N_p^3/3)$
    Compute auxiliary matrix $\mathbf{B}_3 = \mathbf{B}_2(\delta\mathbf{D}s + \delta\mathbf{K})$ $(4nN_p^2)$
    Compute auxiliary matrix $\mathbf{B}_4 = \widetilde{\mathbf{H}}\mathbf{B}_3$ $(4nN_pN_o)$
    Compute $\mathbf{H}_u$ using $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3, \mathbf{B}_4$ in (4.29) $(-)$
    Compute $\mathbf{S_{vv}}$ using (4.18) $(4N_o^2N_f + 4N_oN_f^2)$
  end for
end for
Compute statistics from the generated samples

only the required columns of $\mathbf{H}_u$ rather than computing the entire matrix. Algorithm 4 outlines the procedure of the proposed method to compute the statistics of the uncertain system response.

The total computational cost of the proposed can be found by summing the individual computational costs given in Algorithm 4 as

$$C_{p,H} = N_{freq}\left[\frac{8}{3}n^3 + N_s\left(\frac{8}{3}N_p^3 + 8nN_p^2 + 4nN_oN_p\right)\right] \tag{4.30}$$

$$C_{p,S} = N_{freq}\left[\frac{8}{3}n^3 + N_s\left(\frac{8}{3}N_p^3 + 8nN_p^2 + 4nN_oN_p + 4N_fN_o^2 + 4N_f^2N_o\right)\right] \tag{4.31}$$

where $C_{p,H}$ and $C_{p,S}$ are the computational costs associated with the computation of the transfer function matrix and the PSD of the system's response, respectively. The gains in computational efficiency expected from the proposed method can be computed

from (4.21), (4.22), (4.30) and (4.31) and are given by

$$G_H = \frac{C_{b,H}}{C_{p,H}} = \frac{\frac{8}{3}N_s n^3}{\frac{8}{3}n^3 + N_s\left(\frac{8}{3}N_p^3 + 8nN_p^2 + 4nN_oN_p\right)} \tag{4.32}$$

$$G_S = \frac{C_{b,S}}{C_{p,S}} = \frac{N_s\left(\frac{8}{3}n^3 + 4N_o^2N_f + 4N_oN_f^2\right)}{\frac{8}{3}n^3 + N_s\left(\frac{8}{3}N_p^3 + 8nN_p^2 + 4nN_oN_p + 4N_fN_o^2 + 4N_f^2N_o\right)} \tag{4.33}$$

where G_H and G_S are the expected gains in the computational efficiency associated with the computation of the transfer function matrix and the PSD of the response of the system, respectively. As $N_s \to \infty$, the asymptotic values of the gains in efficiency are given by

$$G_{H,\infty} = \frac{C_{b,H}}{C_{p,H}} = \frac{\frac{8}{3}n^3}{\frac{8}{3}N_p^3 + 8nN_p^2 + 4nN_oN_p} \tag{4.34}$$

$$G_{S,\infty} = \frac{C_{b,S}}{C_{p,S}} = \frac{\frac{8}{3}n^3 + 4N_o^2N_f + 4N_oN_f^2}{\frac{8}{3}N_p^3 + 8nN_p^2 + 4nN_oN_p + 4N_fN_o^2 + 4N_f^2N_o}. \tag{4.35}$$

Since $N_p, N_o \leq n$ and $n^3 \gg n^2$ for large-scale systems, the numerator of (4.35) can be approximated as $8n^3/3$ (same as that in (4.34)).

Until now, the development has dealt with general form of uncertainties that perturb entire rows of $\Delta\mathbf{D}$ and $\Delta\mathbf{K}$ matrices, as discussed earlier. However, strictly diagonal-block perturbations are much more common in civil structures. In such scenarios, the uncertain portions of the damping and the stiffness matrices, $\Delta\mathbf{D}$ and $\Delta\mathbf{K}$, can be written as

$$\Delta\mathbf{D} = \mathbf{L}\delta\mathbf{D}\mathbf{L}^{\mathsf{T}} \tag{4.36}$$

$$\Delta\mathbf{K} = \mathbf{L}\delta\mathbf{K}\mathbf{L}^{\mathsf{T}}. \tag{4.37}$$

In (4.36) and (4.37), $\delta\mathbf{D}$ and $\delta\mathbf{K}$ will be $N_p \times N_p$ matrices. The transfer function matrix of the system is given by

$$\mathbf{H}_u = \left[\mathbf{I}_{n\times n} - \widetilde{\mathbf{H}}[\mathbf{I}_{N_p\times N_p} + (\delta\mathbf{D}s + \delta\mathbf{K})\mathbf{L}^{\mathsf{T}}\widetilde{\mathbf{H}}]^{-1}(\delta\mathbf{D}s + \delta\mathbf{K})\mathbf{L}^{\mathsf{T}}\right]\mathbf{H}. \tag{4.38}$$

The PSD of the system's response can be computed using (4.17) as earlier. The

associated computational costs now become

$$C_{p,H} = N_{freq}\left[\frac{8}{3}n^3 + N_s\left(\frac{20}{3}N_p^3 + 8N_oN_p^2\right)\right] \tag{4.39}$$

$$C_{p,S} = N_{freq}\left[\frac{8}{3}n^3 + N_s\left(\frac{20}{3}N_p^3 + 8N_oN_p^2 4N_fN_o^2 + 4N_f^2N_o\right)\right]. \tag{4.40}$$

The expected gains in computational efficiency can be shown to be

$$G_H = \frac{C_{b,H}}{C_{p,H}} = \frac{\frac{8}{3}N_sn^3}{\frac{8}{3}n^3 + N_s\left(\frac{20}{3}N_p^3 + 8N_oN_p^2\right)} \tag{4.41}$$

$$G_S = \frac{C_{b,S}}{C_{p,S}} = \frac{N_s\left(\frac{8}{3}n^3 + 4N_o^2N_f + 4N_oN_f^2\right)}{\frac{8}{3}n^3 + N_s\left(\frac{20}{3}N_p^3 + 8N_oN_p^2 + 4N_fN_o^2 + 4N_f^2N_o\right)}. \tag{4.42}$$

As $N_s \to \infty$, the asymptotic values of the gains in efficiency are given by

$$G_{H,\infty} = \frac{C_{b,H}}{C_{p,H}} = \frac{\frac{8}{3}n^3}{\frac{20}{3}N_p^3 + 8N_oN_p^2} \tag{4.43}$$

$$G_{S,\infty} = \frac{C_{b,S}}{C_{p,S}} = \frac{\frac{8}{3}n^3 + 4N_o^2N_f + 4N_oN_f^2}{\frac{20}{3}N_p^3 + 8N_oN_p^2 + 4N_fN_o^2 + 4N_f^2N_o}. \tag{4.44}$$

Again, the numerator of (4.35) can be approximated as $8n^3/3$ as discussed earlier. Limiting values of the expected gains in computational efficiencies depend upon the relative magnitude of N_p and N_o. Keeping only the highest-order terms, following asymptotic gains in computational efficiencies can be obtained:

- $N_p \approx N_o$

$$G_{H,\infty} \approx \frac{\frac{8}{3}n^3}{\frac{44}{3}N_p^3} = \frac{2}{11}\left(\frac{n}{N_p}\right)^3 \tag{4.45}$$

$$G_{S,\infty} \approx \frac{\frac{8}{3}n^3}{\frac{44}{3}N_p^3} = \frac{2}{11}\left(\frac{n}{N_p}\right)^3 \tag{4.46}$$

- $N_p \gg N_o$

$$G_{H,\infty} \approx \frac{\frac{8}{3}n^3}{\frac{20}{3}N_p^3} = \frac{2}{5}\left(\frac{n}{N_p}\right)^3 \tag{4.47}$$

$$G_{S,\infty} \approx \frac{\frac{8}{3}n^3}{\frac{20}{3}N_p^3} = \frac{2}{5}\left(\frac{n}{N_p}\right)^3 \tag{4.48}$$

- $N_p \gg N_o$

$$G_{H,\infty} \approx \frac{\frac{8}{3}n^3}{\frac{20}{3}N_p^3 + 8N_oN_p^2} \tag{4.49}$$

$$G_{S,\infty} \approx \frac{\frac{8}{3}n^3}{4N_o^2} = \frac{2N_o}{3}\left(\frac{n}{N_o}\right)^3. \tag{4.50}$$

Appropriate theoretical descriptions of the gains in computational efficiency can be chosen based on the characteristics of the system involved. The usefulness of (4.45)-(4.50) will be demonstrated by means of a numerical example in the next section.

4.3 Numerical Examples

The first example considers a base-excited multi-story shear beam building model. The PSD of the response of the top story is computed by the brute-force method and the proposed method. These are compared to illustrate the exact nature of the proposed method. The second example considers a cantilever beam with part of the beam resting on a Winkler foundation and is used to illustrate the gains obtained in the computational efficiency using the proposed method. The third example considers a base-isolated 3D building model subjected to an earthquake ground motion. The Monte Carlo simulation method is used to estimate the response characteristics of the building. The final example considers the computation of the transfer function matrix of a real structure, the Cedar Avenue bridge. In addition to illustrating the use of the proposed method to compute response characteristics, the final two examples also emphasize the gains in the computational efficiency of the proposed method obtained for large-scale systems. It should be noted that it is assumed that the parameter uncertainty is independent of the load uncertainty in all the examples.

The proposed method was implemented both in C++ and Matlab®, version 7.8.0.347 (R2009a). Results of the first two examples were obtained from the C++ implementation while that of the final two examples were obtained from the Matlab implementation. The first three numerical studies were performed on a Dell 490 Precision workstation, with a processor speed of 3.0 GHz and 8 GB of RAM while the final study utilized a machine equipped with two 2.40 GHz Intel Xeon E5530 quad-core CPUs with 16 GB of RAM.

4.3.1 Example 1: Multi-story Building Model

An isolated 10-story shear-beam building model is considered as shown in Figure 4.1. The stiffness and the damping of the base layer are perturbed from their nominal values by $\Delta k_b = 10$ MN/m, and $\Delta c_b = 2$ MN $\cdot$ s/m. The model is assumed to be subjected to ground acceleration, which is modeled using the stationary Kanai-Tajimi spectrum given by

$$S_{gg}(\omega) = S_0 \frac{1 + 4\zeta_g^2(\omega/\omega_g)^2}{[1 - (\omega/\omega_g)^2]^2 + 4\zeta_g^2(\omega/\omega_g)^2} \tag{4.51}$$

where $S_{gg}(\omega)$ denotes the PSD of the ground acceleration, ζ_g is the ground damping, ω_g is the predominant ground frequency, and S_0 is the intensity of ground acceleration. These values are chosen so as to simulate an intensity equal to that of the N-S component of the 1940 El-Centro earthquake on firm ground conditions ($\zeta_g = 0.6$, $\omega_g = 15.7$ rad/s, and $S_0 = 0.00427$ m^2/s$^3 \cdot$ rad).

The equations of motion of the nominal building model can be written as

$$m_b\ddot{u}_b + c_b\dot{u}_b + k_b u_b = -\, m_b\ddot{z}_g + c\dot{x}_1 + kx_1 \tag{4.52}$$

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{D}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = -\, \mathbf{M}\mathbf{1}(\ddot{u}_b + \ddot{z}_g) \tag{4.53}$$

where, c_b and k_b are the isolation parameters, u_b is the displacement of the base relative to the ground displacement, $\mathbf{x}$ is the vector of floor displacements relative to the base, $\ddot{z}_g$ is the ground acceleration, and $\mathbf{1}$ is a vector with each element equal to unity. (4.52) and (4.53) can be recast into the desired form of (2.1) using simple algebraic manipulations.

The PSD of the response of the top story, as computed by the brute-force method and the proposed method, is shown in Figure 4.2 which shows that the proposed method does indeed produce exact results. The $2-norm$ of the difference between the PSDs of response of the top story computed using the brute-method and the proposed method is 5.6288×10^{-14}, which is of the order of the machine epsilon and hence, can be attributed to round-off errors in the floating point operations and the numerical condition of the pertinent matrices.

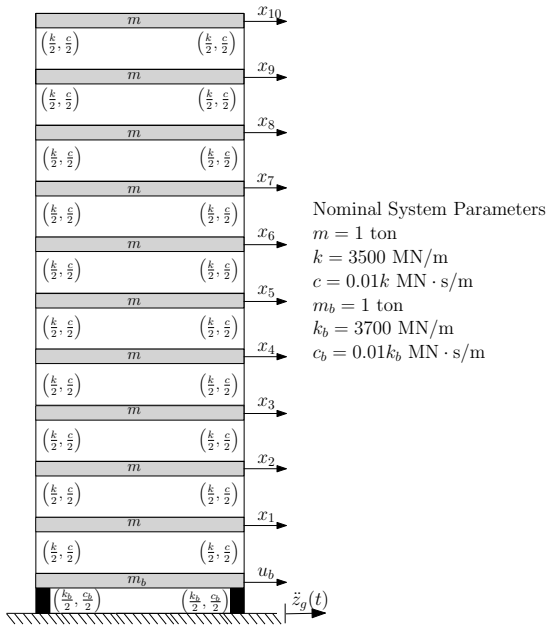


Figure 4.1: Example 1: Multi-storied Building

4.3.2 Example 2: Cantilever Beam Model

A cantilever beam partially supported on a Winkler foundation is considered as shown in Figure 3.4. The mass and the stiffness matrices for the beam are generated using a finite element formulation using standard 2-dimensional Eüler-Bernoulli beam elements. The beam is discretized with 101 nodes, which corresponds to $n = 200$ active degrees-of-freedom. Rayleigh damping is assumed with $\mathbf{D} = \alpha\mathbf{M} + \beta\mathbf{K}$, $\alpha = 0.04309$, and $\beta = 0.009708$. The nominal system for this example is simply the cantilever beam without any foundation. The equations of motion for the nominal system can be written as in (2.1) and that for the uncertain system can be written as in (2.54).

The stiffness of the individual springs used to model the Winkler foundation is assumed to be uncertain, with a uniform distribution between $3EI/L^3$ and $12EI/L^3$. The uncertainty in the foundation damping comes indirectly from that in the foundation stiffness. In terms of prescribing the $\Delta\mathbf{K}$ and $\Delta\mathbf{D}$ matrices, the uncertain Winkler foundation translates to adding nodal springs of uncertain stiffness to the beam which, in turn, corresponds to adding uncertain terms to the elements of the main diagonal of

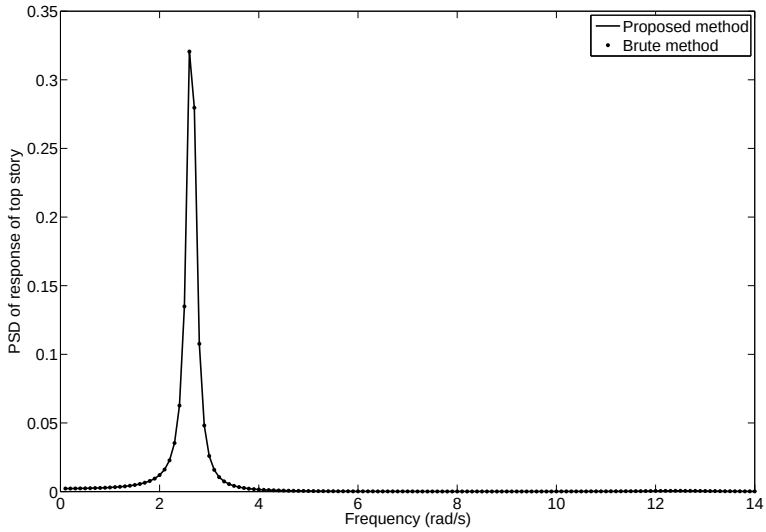


Figure 4.2: PSD of the Response the of Top Story of the Building

both the system stiffness and the damping matrices, respectively.

The input force, $\mathbf{f}(t)$, is a vertically downwards tip-load, assumed to be an AR(1) process with its PSD given by

$$S_{ff}(\omega) = \frac{S_0}{1 + a - a\cos(\omega)} \tag{4.54}$$

where $S_0 = 200$ is the intensity of the load and $a = 0.7$ is the autoregressive filter coefficient.

This example was used to investigate the computational efficiency of the proposed method. Various cases were considered to test the efficiency of the proposed method under different amounts of uncertainty present in the system. The effect of the number of degrees-of-freedom whose output is desired was also studied. To vary the amount of uncertainty, the length of the beam supported on the Winkler foundation, l, was increased from 1% to 39% of the length of the beam. The number of output degrees-of-freedom was varied by selecting equidistant points on the beam whose output was computed. In terms of the nomenclature used in the expected efficiency gain functions, N_p was varied from 1% to 20% of n and N_o was varied from 1% to 100% of n.

The gains in computational efficiency are presented in Tables 4.1-4.3. Figure 4.3 shows

Table 4.1: Gains in Computational Efficiency for Example 2
($N_o = 1\%$ of n)

$\mathbf{N_s} \Downarrow$ $\mathbf{N_p}$(% of $\mathbf{n}$) $\Rightarrow$	**1%**	**5%**	**10%**	**20%**
1	0.96	0.98	0.97	0.94
10	9.72	9.25	9.20	7.00
100	96.47	89.23	61.96	20.14
1000	927.02	496.57	144.28	24.69
10000	6643.75	914.08	167.79	26.45

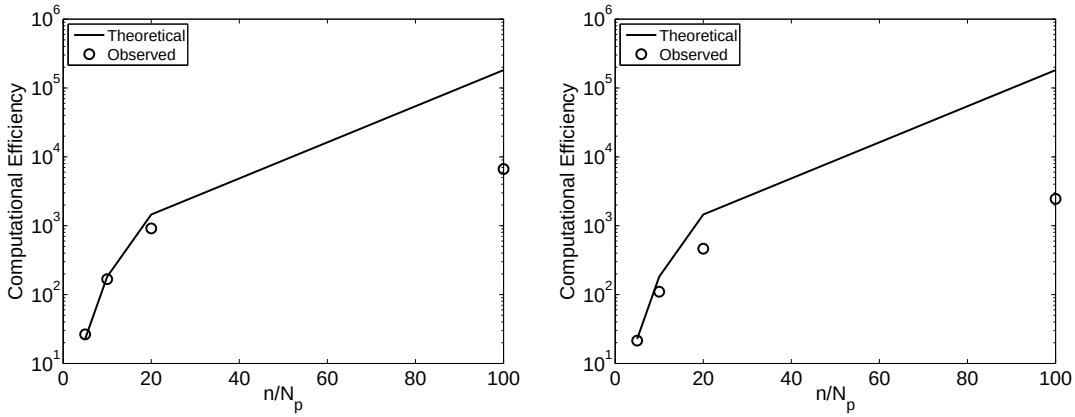
Table 4.2: Gains in Computational Efficiency for Example 2
($N_o = 20\%$ of n)

$\mathbf{N_s} \Downarrow$ $\mathbf{N_p}$(% of $\mathbf{n}$) $\Rightarrow$	**1%**	**5%**	**10%**	**20%**
1	0.99	0.97	0.96	0.93
10	9.79	9.52	8.94	6.55
100	99.12	81.15	52.09	16.53
1000	783.04	324.70	99.78	19.51
10000	2452.65	463.12	110.05	21.43

Table 4.3: Gains in Computational Efficiency for Example 2
($N_o = 100\%$ of n)

$\mathbf{N_s} \Downarrow$ $\mathbf{N_p}$(% of $\mathbf{n}$) $\Rightarrow$	**1%**	**5%**	**10%**	**20%**
1	0.98	0.86	0.94	0.86
10	8.26	3.68	6.05	3.68
100	33.34	5.57	15.52	5.57
1000	48.06	34.30	18.13	5.87
10000	49.96	35.29	19.10	5.95

the observed gains in computational efficiency compared to those theoretically predicted by (4.45)-(4.50) for $N_s = 10,000$. It can be observed that the theoretical expressions given in (4.45)-(4.50) accurately predict the expected gains in computational efficiency for smaller problem sizes. For larger problem sizes, the deviation of the theoretical



(a) $N_o = 1\%$ (b) $N_o = 20\%$

Figure 4.3: Theoretical vs Observed Gains in Computational Efficiency for Example 2

predictions can be attributed to data latency caused due to the large size of data structures involved that cannot be accommodated entirely within the processor's cache memory.

It is evident that the proposed method is computationally very efficient, yielding $\mathcal{O}\left(10^3\right)$ efficiency gains compared to the brute-force method for small values of N_p and N_o. Even when all the degrees-of-freedom are required, which is seldom the case in real applications, the proposed method offers an $\mathcal{O}\left(10^1\right)$ gain. Further, it can be noticed that as the size of the system uncertainty (N_p) is increased, the computational efficiency of the proposed method decreases, which is a direct consequence of the fact that the principal component of the efficiency of the proposed method is the inversion of an $N_p \times N_p$ matrix instead of a full $n \times n$ matrix.

4.3.3 Example 3: 3D Building Model with Base Isolation Layer

A 615 degree-of-freedom three-dimensional steel building, as shown in Figure 3.6, is considered to illustrate the computation of stochastic response characteristics using the proposed method. The building is modeled using a finite element formulation and is assumed to be isolated from the ground through a bed of translational and rotational springs and a rigid block mass, M_b. This isolation bed is subsequently

lumped into three springs and dashpots at the center of mass of the base block. The FE model of the building itself contributes 612 degrees-of-freedom, while the remaining 3 degrees-of-freedom are contributed by the rigid mass (two translations and one rotation). Rayleigh damping is assumed for both the building and the rigid mass. The proportionality coefficients of the damping model for the building are $\alpha_{\text{building}} = 0.4507$ and $\beta_{\text{building}} = 0.0011$ and for the rigid mass are $\alpha_{\text{block}} = 0$ and $\beta_{\text{block}} = 0.002$.

The stiffnesses of the translational and the rotational springs that connect the rigid mass to the ground are assumed to be independent random variables as described in Figure 3.6. The uncertainty in the damping is induced indirectly by the uncertainty in the stiffness by employing the same relationship as for the nominal system. A Kanai-Tajimi spectrum is used to model the input ground acceleration, identical from the first example.

To compute the statistical characteristics of the response of the building, $10,000$ simulations of the response of the building were computed using the proposed method. Figure 4.4 shows five samples of the PSD of the relative base drift of the model out of the $10,000$ samples that were generated. It can be seen that the first mode of the system shifts significantly because of the uncertainties in the isolation springs. Figure 4.5 shows the mean PSD of the relative base drift of the uncertain system, with $N_s = 10,000$, along with the PSD of the relative base drift of the nominal system. In addition to reiterating the fact that the fundamental frequency of the system shifts, the PSD also shows that the average energy of the system response spreads over a wide frequency range instead of being confined to a narrow-band as for the nominal system. The limits on the y-axis are curtailed to depict this behavior clearly. In the inset, full PSD of the base drift of the nominal system is presented to show that the system is stable.

In order to account for all significant natural frequencies, and obtain a better quantitative estimate of the system's response, the frequency resolution and the range are increased for this example. The time of run for the brute-force method for the generation of the required $10,000$ samples is projected from the computational time required for 1 iteration, and that for the proposed method is the actual computational time. The estimated time to generate $10,000$ samples of the output PSD using the brute-force method is 588.17×10^4 seconds ($\sim$ 68 days), and that using the proposed method is 1375.70 seconds ($\sim$ 23 minutes). This corresponds to an efficiency gain of

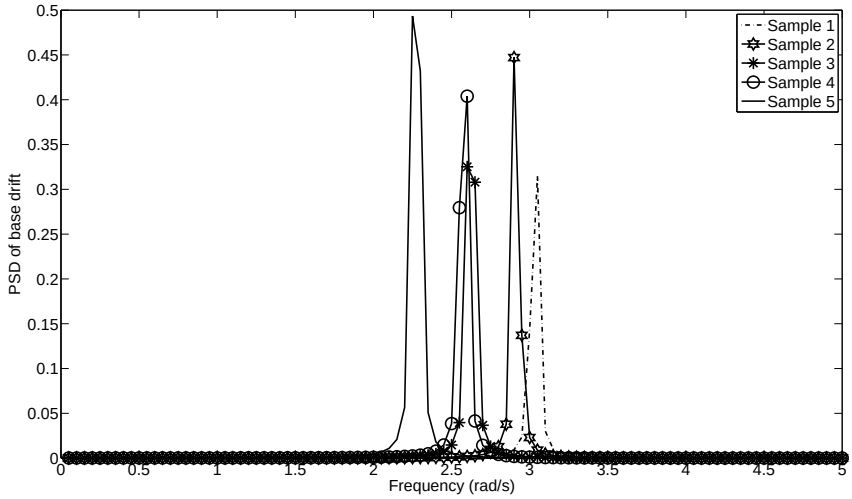


Figure 4.4: Sample Realizations of PSD of Relative Base Drift

about $4,275.42$ times.

4.3.4 Example 4: Cedar Avenue Bridge Model

The final example considers the computational model of an actual bridge, the Cedar Avenue bridge, in order to demonstrate the computational efficiency of the proposed method when applied to a real life system. The bridge is located in the metro area of the Twin Cities in Minnesota. Thompson and Schultz [204] created a finite element model of the Cedar Avenue Bridge in SAP2000 [205] using 3D frame and shell elements. Figure 3.8 shows a photograph and the finite element model of the Cedar Avenue bridge.

The bridge model is equipped with a scissor jack apparatus as shown in Figure 3.9(a) to provide counter moments in order to reduce the moment range experienced by certain joints. Gastineau *et al.* [206] developed the finite element model of the modified bridge consisting of $17,687$ degrees-of-freedom as shown in Figure 3.9(b). The modification apparatus is connected across four joints, thereby causing a rank-8 perturbation in the system's stiffness matrix. The stiffness properties of the modification apparatus are assumed to be uncertain, modeled using a uniform distribution within a 4% range of the corresponding nominal values.

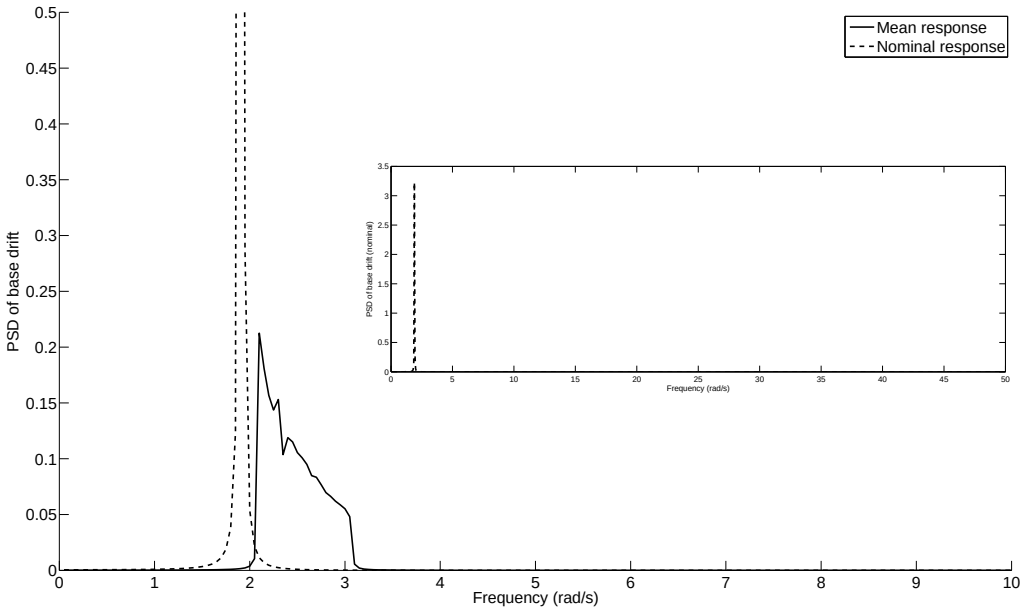


Figure 4.5: Mean and Nominal PSD of Relative Base Drift

Table 4.4 shows the required computational times and gains in computational efficiency for $1,000$ and $10,000$ sample realizations of the transfer function matrix of the system. The proposed method achieves $\mathcal{O}\left(10^2\right)$-$\mathcal{O}\left(10^3\right)$ gain in computational efficiency when $10,000$ sample realizations of the transfer function matrix of the system are computed. The full transfer function matrix of the system has been computed approximately using its first 500 smallest magnitude eigenvalues and corresponding eigenvectors for both the brute-force and the proposed method.

4.4 Conclusions

This chapter presented the development of an efficient computational methodology for the determination of the transfer function matrix of LTI systems with local damping and stiffness uncertainties. The proposed method, outlined in Algorithm 4, utilized the frequency response of an underlying nominal system to compute the response of

Table 4.4: Computational Times and Efficiency Gain Ratios for Example 4

	1,000 samples		**10,000 samples**	
# DOFs computed⇒	2	10	2	10
Brute-force method CPU time (t_b)	1.2 months	1.2 months	12 months	12 months
Proposed method CPU time (t_p)	81 minutes	2 hours	5.6 hours	13.8 hours
Computational efficiency (t_b/t_p)	640	405	1534	625

a family of related systems. The principal component of the computational efficiency of the proposed method was the reduction of the size of matrices being inverted from $n \times n$ to $N_p \times N_p$. Some characteristics of the proposed method that make it a versatile procedure that can be utilized in many application scenarios include:

- ability to accommodate non-symmetric system matrices
- ability to accommodate non-symmetric perturbations in matrices
- ability to accommodate non-classical damping
- ability to directly compute the output of only selected degrees-of-freedom
- absence of any approximations in the analytical development

Four examples were presented to highlight various aspects of the proposed method. The first example demonstrated the exact nature of the proposed method. The magnitude of the errors were found to be of the order of the machine epsilon, which can be attributed to the truncations involved in finite arithmetic and conditioning of the pertinent matrices. This exactness was also verified for the other examples.

The second example demonstrated the computational efficiency of the proposed method for various combinations of N_p, the size of system uncertainty, and N_o, the number of outputs computed. The results also showed the usefulness of the theoretical computational efficiency gains computed in Section 4.2. These theoretical expressions appropriately described the observed gains in computational efficiency.

The third example concerned the application of the proposed method to a moderate-sized computational model with local damping and stiffness uncertainties. Computational gains of $\mathcal{O}\left(10^3\right)$ were observed. The final example demonstrated the application of the proposed method to the computational model of a real structure; gains in computational efficiency of $\mathcal{O}\left(10^2\right)$-$\mathcal{O}\left(10^3\right)$ were observed.

Chapter 5

Time Domain Response of Systems

This chapter discusses the development of a computationally efficient method to analyze time-invariant dynamical systems with localized nonlinearities and/or uncertainties in the time domain. The proposed method utilizes a nonlinear Volterra integral equation (NVIE) based solution procedure; a Newton-Gregory quadrature scheme is utilized to discretize the governing system of NVIEs. The proposed method is (i) able to directly compute the response of only a selected subset of the full response, (ii) able to handle both smooth and non-smooth nonlinearities, and (iii) demonstrates good computational efficiency so that a large number of samples can be computed for use in uncertainty quantification procedures. Moreover, the proposed method has been developed in state-space and can thus be applied to a wide range of dynamical systems.

The equations of motion of an $\underline{n}$-dimensional LTI system can be written in the time domain as

$$\dot{\underline{\mathbf{x}}}(t) = \mathbf{A}\,\underline{\mathbf{x}}(t) + \underline{\mathbf{B}}\,\mathbf{f}(t), \quad \underline{\mathbf{x}}(t_0) = \underline{\mathbf{x}}_0 \tag{5.1}$$

where $\mathbf{A} \in \mathbb{R}^{\underline{n}\times\underline{n}}$ is the system matrix, $\underline{\mathbf{B}} \in \mathbb{R}^{\underline{n}\times N_f}$ is an influence matrix, $\underline{\mathbf{x}} \in \mathbb{R}^{\underline{n}}$ is the state-vector, and $\mathbf{f} \in \mathbb{R}^{N_f}$ is the input vector. The system given by (5.1) will be referred to as the 'nominal' system in the remainder of this chapter. The time domain solution, $\underline{\mathbf{x}}(t)$, of (5.1) can be written using the convolution integral as

$$\underline{\mathbf{x}}(t) = \underline{\mathbf{h}}(t-t_0)\underline{\mathbf{x}}_0 + \int_{t_0}^{t} \underline{\mathbf{h}}(t-\tau)\mathbf{B}\underline{\mathbf{f}}(\tau)\mathrm{d}\tau \tag{5.2}$$

where $\underline{\mathbf{h}}(t) \in \mathbb{R}^{\underline{n}\times\underline{n}}$ is the impulse response matrix of the system given by

$$\underline{\mathbf{h}}(t) = e^{\mathbf{A}t}. \tag{5.3}$$

The equations of motion of a time-invariant dynamical system with local uncertainties and nonlinearities can be written as

$$\dot{\underline{\mathbf{v}}}(t) = \mathbf{A}\underline{\mathbf{v}}(t) + \mathbf{B}\underline{\mathbf{f}}(t) + \mathbf{L}\underline{\mathbf{u}}(\widehat{\underline{\mathbf{v}}}), \quad \underline{\mathbf{v}}(t_0) = \underline{\mathbf{v}}_0 \tag{5.4}$$

where $\underline{\mathbf{u}}(\widehat{\underline{\mathbf{v}}}) \in \mathbb{R}^{N_p}$ is a nonlinear and/or uncertain vector function that prescribes the nonlinear relationships and the uncertainties of the system; $\mathbf{L} \in \mathbb{R}^{\underline{n}\times N_p}$ is an influence matrix that maps the nonlinearities and the uncertainties to appropriate degrees-of-freedom of the system; $\widehat{\underline{\mathbf{v}}} = \mathbf{L}_1\underline{\mathbf{v}} \in \mathbb{R}^{N_k}$ is the reduced state-vector and represents only those states (or a linear combination of the states) on which the function $\underline{\mathbf{u}}$ depends and $\mathbf{L}_1 \in \mathbb{R}^{N_k\times\underline{n}}$ is another influence matrix used to obtain the reduced state-vector from the full state-vector. The systems of interest have the key characteristics that

$$N_k \ll \underline{n} \qquad \text{and} \qquad N_p \ll \underline{n} \tag{5.5}$$

due to the spatial locality of the nonlinearities and the uncertainties.

The next section will discuss a conventional method to compute the statistics of the relevant components of the time domain response of (5.4). It will be referred to as the 'brute-force method' in the remainder of the chapter. The subsequent section will develop the new approach that can be utilized to compute the response of (5.4) in a computationally efficient manner. Finally, numerical examples will be presented to demonstrate the error behavior and the computational efficiency of the proposed method as compared to the brute-force method.

5.1 Brute-Force Method

Several methods can be utilized to perform the numerical integration of the system of ODEs in (5.4) to obtain the time domain response of the system. An explicit fourth-order Rünge-Kutta method will be employed here. A sampling-based approach will be applied to perform the uncertainty analysis of the system's response. If N_s

samples of the state-vector, $\underline{\mathbf{v}}$, are computed, the mean and the covariance response of the system can be computed using the following unbiased estimators:

$$\boldsymbol{\mu}_{\underline{\mathbf{v}}}(t) = \frac{1}{N_s}\sum_{j=1}^{N_s}\underline{\mathbf{v}}^{(j)}(t) \tag{5.6}$$

$$\boldsymbol{\Gamma}_{\underline{\mathbf{v}}\underline{\mathbf{v}}}(t) = \frac{1}{N_s-1}\sum_{j=1}^{N_s}\left(\underline{\mathbf{v}}^{(j)}(t)-\boldsymbol{\mu}_{\underline{\mathbf{v}}}(t)\right)\left(\underline{\mathbf{v}}^{(j)}(t)-\boldsymbol{\mu}_{\underline{\mathbf{v}}}(t)\right)^{\mathsf{T}}. \tag{5.7}$$

Algorithm 5 outlines the brute-force procedure to compute the statistics of the time domain response of the system. The total computational cost associated with computing

Algorithm 5 Brute-Force Method to Compute Time Domain Response of (5.4)

Require: $\mathbf{A}$, $\mathbf{B}$, $\mathbf{L}$, $\underline{\mathbf{u}}$, $\mathbf{L}_1$

```
Generate N_{s_f} samples of the load, f
Generate N_{s_p} samples of the uncertain parameters in v̂
N_s = N_{s_f} N_{s_p}
for all samples of f do
    for all samples of v̂ do
        Solve (5.4) using explicit fourth-order Rünge-Kutta method        (4 N_t n^2)
    end for
end for
Compute required statistics using (5.6) and (5.7)
```

all N_s samples of the time domain response of the system will be

$$C_b = 4N_sN_t\underline{n}^2. \tag{5.8}$$

5.2 Proposed Method

The response of (5.4) can be written in terms of the response of the nominal system, (5.1), using the superposition principle as

$$\underline{\mathbf{v}}(t) = \underline{\mathbf{x}}(t) + \int_{t_0}^{t}\underline{\mathbf{h}}_L(t-\tau)\underline{\mathbf{u}}\left(\underline{\widehat{\mathbf{v}}}(\tau)\right)\mathrm{d}\tau \tag{5.9}$$

where $\underline{\mathbf{x}}(t)$ is the response of the nominal system subjected to the given input, $\mathbf{f}(t)$, and $\underline{\mathbf{h}}_L(t)$ is the response of the nominal system subjected to a unit impulse applied in

the pattern of $\mathbf{L}$; *i.e.*, if $\delta(t)$ is the Dirac's delta function, $\underline{\mathbf{h}}_L(t)$ is the response of the following system of ODEs:

$$\dot{\underline{\mathbf{h}}}_L(t) = \mathbf{A}\underline{\mathbf{h}}_L(t) + \mathbf{L}\delta(t). \tag{5.10}$$

(5.9) is a system of $\underline{n}$ NVIEs of the second kind. The dimensionality of the system of NVIEs in (5.9) can be reduced by recognizing the fact that the kernel of the system of NVIEs depends only on the N_k-dimensional reduced state-vector, $\underline{\widehat{\mathbf{v}}}$, rather than the full $\underline{n}$-dimensional state-vector. To achieve this dimension reduction, an unknown function, $\mathbf{p}(t)$, can be defined as

$$\mathbf{p}(t) = \underline{\mathbf{u}}\left(\underline{\widehat{\mathbf{v}}}(t)\right) \tag{5.11}$$

which is the force exerted by some nonlinear and/or stochastic element of the system.

Since the function $\underline{\mathbf{u}}$ depends only on the reduced state-vector, $\underline{\widehat{\mathbf{v}}}$, (5.9) can be rewritten for the N_k states present in $\underline{\widehat{\mathbf{v}}}$ using (5.11) as

$$\underline{\widehat{\mathbf{v}}}(t) = \underline{\widehat{\mathbf{x}}}(t) + \int_{t_0}^{t} \underline{\widehat{\mathbf{h}}}_L(t-\tau)\mathbf{p}(\tau)\mathrm{d}\tau \tag{5.12}$$

where $\underline{\widehat{\mathbf{x}}} = \mathbf{L}_1\underline{\mathbf{x}} \in \mathbb{R}^{N_k}$ is the required subset (or linear combinations) of the response of the nominal system and $\underline{\widehat{\mathbf{h}}}_L = \mathbf{L}_1\underline{\mathbf{h}}_L$ is the required portion of the corresponding impulse response matrix of the linear system. Substituting (5.12) in (5.11) yields

$$\mathbf{p}(t) - \underline{\mathbf{u}}\left(\underline{\widehat{\mathbf{x}}}(t) + \int_{t_0}^{t} \underline{\widehat{\mathbf{h}}}_L(t-\tau)\mathbf{p}(\tau)\mathrm{d}\tau\right) = \mathbf{0}_{N_k\times 1} \tag{5.13}$$

which is a system of N_k NVIEs written in a non-standard form and which can be solved to obtain the unknown force, $\mathbf{p}$. Once $\mathbf{p}$ is known, the desired states of the nonlinear system can be computed subsequently as

$$\mathbf{y}(t) = \mathbf{C}\underline{\mathbf{v}} = \mathbf{C}\left[\underline{\mathbf{x}}(t) + \int_{0}^{t} \underline{\mathbf{h}}_L(t-\tau)\mathbf{p}(\tau)\mathrm{d}\tau\right] \tag{5.14}$$

where $\mathbf{C}$ is an output matrix. The non-standard form of the NVIEs achieves two purposes: (i) it reduces the size of the system of NVIEs from $\underline{n}$ to N_k and (ii) it allows for the computation of only the desired states of the nonlinear system by simple convolution rather than solving a system of NVIEs.

As there are no analytical solutions of (5.13), one must resort to numerical techniques utilizing a time discretization. Let subscript k on a symbol denote the corresponding

continuous function evaluated at a time t_k, *i.e.*, $\underline{\mathbf{x}}_k = \underline{\mathbf{x}}(t_k)$, $\underline{\mathbf{v}}_k = \underline{\mathbf{v}}(t_k)$, $\underline{\mathbf{h}}_{L_k} = \underline{\mathbf{h}}_L(t_k)$, and $\mathbf{p}_k = \mathbf{p}(t_k)$. Using a Newton-Gregory integration scheme with constant time-step, Δt, one can discretize (5.12) as

$$\widehat{\underline{\mathbf{v}}}_k = \widehat{\underline{\mathbf{x}}}_k + \Delta t \sum_{i=0}^{k} w_{i,k} \widehat{\underline{\mathbf{h}}}_{L_{k-i}} \mathbf{p}_i \tag{5.15}$$

where $w_{i,k}$ are the weights associated with the Newton-Gregory quadrature rule employed. Welsch [207] presented an algorithm that can compute weights for an arbitrary order Newton-Gregory quadrature rule (however, as will be demonstrated in the numerical studies presented later in this chapter, quadrature rules of order six or higher can sometimes demonstrate erratic convergence rates due to the higher-order interpolations involved). The weights for quadrature rules, given in Table 5.1 up to order six, are symmetric (*i.e.*, $w_{i,k} = w_{k-i,k}$) and all but the first and last $r = (k_{\min} + 1)/2$ weights are unity (*i.e.*, $w_{i,k} = 1$ for $i \in [r, k-r]$). $k_{\min}$, which increases as a function of the quadrature order, is the minimum number of steps necessary to use the quadrature rule. An integration of shorter duration must be computed using another appropriate solution procedure of the same order of accuracy as the quadrature rule; herein, a matching-order Rünge-Kutta scheme has been utilized to accomplish this.

Table 5.1: Weights for Different Order Quadrature Rules

2nd Order Rule (trapezoidal rule)
$k_{min} = 1,\ r = 1$ $w_{0,k} = w_{k,k} = \frac{1}{2}$ $w_{i,k} = 1; \quad i = r, \ldots, (n-r).$
4th Order Rule
$k_{min} = 5,\ r = 3$ $w_{0,k} = w_{k,k} = \frac{3}{8},\ w_{1,k} = w_{k-1,k} = \frac{7}{6},\ w_{2,k} = w_{k-2,k} = \frac{23}{24}$ $w_{i,k} = 1; \quad i = r, \ldots, (k-r).$
6th Order Rule
$k_{min} = 9,\ r = 5$ $w_{0,k} = w_{k,k} = \frac{95}{288},\ w_{1,k} = w_{k-1,k} = \frac{317}{240},\ w_{2,k} = w_{k-2,k} = \frac{23}{30},$ $w_{3,k} = w_{k-3,k} = \frac{793}{720},\ w_{4,k} = w_{k-4,k} = \frac{157}{160},$ $w_{i,k} = 1; \quad i = r, \ldots, (k-r).$

Although (5.15) can be used directly to solve for the unknown force, $\mathbf{p}(t)$, it is

advantageous from a computational point of view to eliminate the weights, $w_{i,k}$, so that a standard convolution-type formulation is obtained. This can be easily accomplished by exploiting the symmetry of the weights and the fact that majority of the weights are unity. (5.15) can be rewritten as

$$\underline{\widehat{\mathbf{v}}}_k = \underline{\widehat{\mathbf{x}}}_k + \Delta t \sum_{i=0}^{k} \underline{\widetilde{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i \tag{5.16}$$

where $\underline{\widetilde{\mathbf{g}}} = \mathbf{L}_1 \underline{\mathbf{g}}$ and where $\underline{\mathbf{g}}$ and $\widetilde{\mathbf{p}}$ are same as $\underline{\mathbf{h}}_L$ and $\mathbf{p}$ except that the first r terms of the time-histories of $\underline{\mathbf{h}}_L$ and $\mathbf{p}$ are multiplied by $w_{i,k}$; *i.e.*, $\underline{\mathbf{g}}_i = w_{i,k} \underline{\mathbf{h}}_{L_i}$ and $\widetilde{\mathbf{p}}_i = w_{i,k} \mathbf{p}_i$ for $i \in [0, r-1]$, and $\underline{\mathbf{g}}_i = \underline{\mathbf{h}}_{L_i}$ and $\widetilde{\mathbf{p}}_i = \mathbf{p}_i$ for $i \geq r$. Thus, the discretized form of the system of NVIEs, (5.13), can be obtained as

$$\mathbf{p}_k - \underline{\mathbf{u}} \left(\underline{\widehat{\mathbf{x}}}_k + \Delta t \sum_{i=0}^{k} \underline{\widetilde{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i \right) = \mathbf{0}_{N_k \times 1}. \tag{5.17}$$

This system of nonlinear algebraic equations in $\mathbf{p}_k$ can be solved at each time-step using a Newton's method with a desired error tolerance. With $\mathbf{p}_k^j$ denoting the estimate of $\mathbf{p}_k$ in the jth iteration of Newton's method, initial conditions $\mathbf{p}_0 = \underline{\mathbf{u}} \left(\underline{\widehat{\mathbf{v}}}(0) \right) = \underline{\mathbf{u}} \left(\mathbf{L}_1 \underline{\mathbf{v}}_0 \right)$, and $\mathbf{p}_k^0 = \mathbf{p}_{k-1}$ (*i.e.*, using the previous time step as the initial estimate of the next time step), the update equation of the Newton's method can be written as

$$\mathbf{p}_k^{j+1} = \mathbf{p}_k^j - \left[\mathbf{I} - \frac{\partial \underline{\mathbf{u}}}{\partial \underline{\widehat{\mathbf{v}}}} \underline{\widetilde{\mathbf{g}}}_0 \Delta t \right]^{-1} \left[\mathbf{p}_k^j - \underline{\mathbf{u}} \left(\underline{\widehat{\mathbf{x}}}_k + \Delta t \sum_{i=0}^{k} \underline{\widetilde{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i \right) \right]. \tag{5.18}$$

In all the examples considered later in this chapter, an error tolerance of $\mathcal{O}\left(10^{-12}\right)$ was prescribed, which was typically attained in about 5 iterations. Once $\mathbf{p}$ is known, the response of the desired states of the system can be directly computed using the convolution (5.14) by any appropriate manner such as by the same discretization and quadrature used for the states

$$\mathbf{y}_k = \mathbf{C} \underline{\mathbf{v}}_k = \mathbf{C} \left[\underline{\mathbf{x}}_k + \Delta t \sum_{i=0}^{k} \underline{\mathbf{g}}_{k-i} \widetilde{\mathbf{p}}_k \right]. \tag{5.19}$$

It can be observed that the majority of the computational cost associated in solving (5.17) comes from evaluating the convolution. If the convolution is computed in a straightforward manner, the solution of (5.17) for N_t time-steps requires the convolution

of two vectors of length N_t which will require $\mathcal{O}\left(N_t^2\right)$ operations (multiplications and additions). It is well known that such a convolution can be computed in a much more efficient manner, in only $\mathcal{O}\left(N_t \log N_t\right)$ operations, using the FFT [208]. However, to compute the convolution using the FFT, both the sequences being convolved must be completely known *a priori*. Since this requirement is not fulfilled when solving a Volterra equation step-by-step in time, a different approach must be taken. The convolution space can be sub-divided into various blocks of squares and triangles, as shown in Figure 5.1. Such a division of the convolution space allows the use of the FFT to compute portions of the convolution, as described in the subsequent paragraphs.

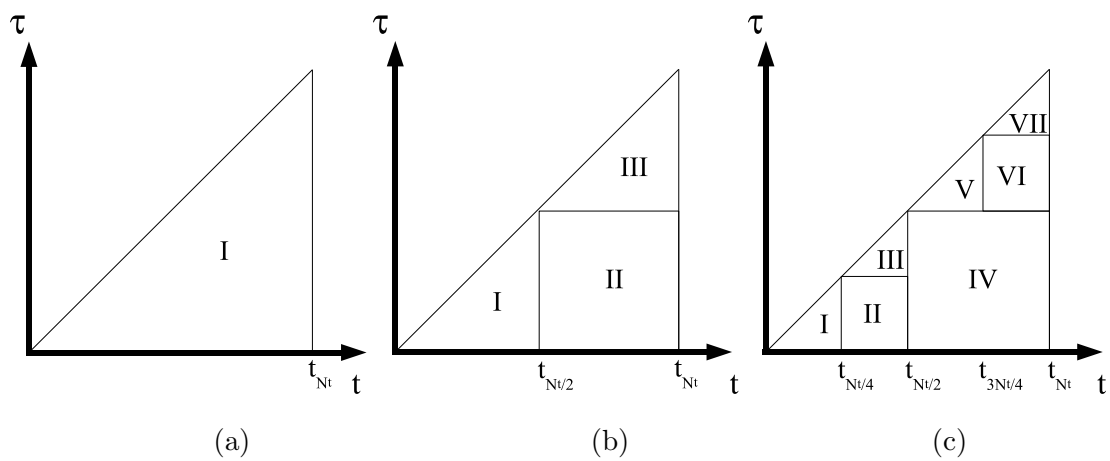


Figure 5.1: Sub-division of the Convolution Space
(Squares by FFT and Triangles by Quadrature Rule)

Let $N_t = 2^l$ denote the total number of time points for some positive integer l. The assumption that N_t is a whole power of 2 is crucial for an efficient implementation of the FFT routine. Figure 5.1(a) depicts a naïve implementation of the convolution. The solution proceeds via the following equation:

$$\mathbf{p}_k - \underline{\mathbf{u}}\left(\underline{\widehat{\mathbf{x}}}_k + \Delta t \sum_{i=0}^{k} \underline{\widetilde{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i\right) = 0; \qquad k = 0, \ldots, N_t - 1. \tag{5.20}$$

This procedure does not utilize the FFT to compute the convolution and hence, requires $\mathcal{O}\left(N_k^2 N_t^2\right)$ operations. However, if the convolution space is partitioned in half, as shown in Figure 5.1(b), then a more efficient solution proceeds via the following steps:

Step 1: Compute $\mathbf{p}_k$ for $k = 0, \ldots, N_t/2 - 1$ using

$$\mathbf{p}_k - \underline{\mathbf{u}} \left(\widehat{\underline{\mathbf{x}}}_k + \Delta t \sum_{i=0}^{k} \widetilde{\underline{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i \right) = 0; \qquad k = 0, \ldots, N_t/2 - 1 \tag{5.21}$$

which requires $\mathcal{O}\left(N_k^2 \frac{N_t^2}{4}\right)$ operations.

Step 2: Compute $\mathbf{p}_k$ for $k = N_t/2, \ldots, N_t - 1$ using

$$\mathbf{p}_k - \underline{\mathbf{u}} \left(\widehat{\underline{\mathbf{x}}}_k + \Delta t \sum_{i=0}^{k} \widetilde{\underline{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i \right) = 0; \qquad k = N_t/2, \ldots, N_t - 1. \tag{5.22}$$

Since $\widetilde{\mathbf{p}}_j$ is now known for $j = 0, \ldots, N_t/2 - 1$, the convolution in (5.22) can be broken into two terms as

$$\Delta t \sum_{i=0}^{k} \widetilde{\underline{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i = \Delta t \sum_{i=0}^{N_t/2-1} \widetilde{\underline{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i + \Delta t \sum_{i=N_t/2}^{k} \widetilde{\underline{\mathbf{g}}}_{k-i} \widetilde{\mathbf{p}}_i \tag{5.23}$$

The first term in (5.23) can be computed using the FFT and thus requires $\mathcal{O}\left(N_k^2 \frac{N_t}{2} \log \frac{N_t}{2}\right)$ operations. The second term in (5.23) is computed in a manner similar to Step 1, requiring $\mathcal{O}\left(N_k^2 \frac{N_t^2}{4}\right)$ operations.

Thus, in this implementation, the total number of operations required is reduced to $\mathcal{O}\left(N_k^2 \frac{N_t^2}{2} + N_k^2 \frac{N_t}{2} \log \frac{N_t}{2}\right)$.

This idea can be extended in a similar fashion with a greater number of partitions. For instance, if the convolution space is further halved, as shown in Figure 5.1(c), the number of operations required will be

$$\mathcal{O}\left(N_k^2 \frac{N_t^2}{4} + N_k^2 \frac{N_t}{2} \log \frac{N_t}{4} + N_k^2 \frac{N_t}{2} \log \frac{N_t}{2}\right).$$

In the limiting case, the number of operations required for the convolution can be shown to be $\mathcal{O}\left(N_k^2 N_t \log^2 N_t\right)$. Algorithm 6 outlines the brute-force procedure to compute the statistics of the time domain response of the system. The total computational cost associated with computing all N_s samples of the time domain response of the system will be

$$C_p = 4 N_t \underline{n}^2 + N_{s_f} N_N N_k^2 N_t \log^2 N_t + N_s N_o^2 N_t \log N_t \tag{5.24}$$

where N_N is the number of Newton iterations required to solve (5.18) and N_o is the number of states computed.

Algorithm 6 Proposed Method to Compute Time Domain Response of (5.4)

Require: $\mathbf{A}$, $\mathbf{B}$, $\mathbf{L}$, $\underline{\mathbf{u}}$, $\mathbf{L}_1$

Generate N_{s_f} samples of the load, $\mathbf{f}$
Generate N_{s_p} samples of the uncertain parameters in $\underline{\widehat{\mathbf{v}}}$
$N_s = N_{s_f} N_{s_p}$
for all samples of $\mathbf{f}$ **do**
  Solve (5.1) using explicit fourth-order Rünge-Kutta method $(4N_t\underline{n}^2)$
  for all samples of $\underline{\widehat{\mathbf{v}}}$ **do**
    Solve (5.17) using the proposed method $(N_k^2 N_t \log^2 N_t)$
    Compute required states using (5.19) $(N_o^2 N_t \log N_t)$
  end for
end for
Compute required statistics using (5.6) and (5.7)

5.3 Numerical Examples

This section demonstrates the method developed in the preceding section through four numerical examples. The first example considers a multi-story shear beam building model with a linear damping modification and will be used to examine the convergence properties and the effect of number of partitions used in the convolution on the proposed method. The second example demonstrates the application of the method to a system with multiple nonlinearities by considering two shear beam building models coupled to each other. The third example considers a 3D building model and demonstrates the scaling of the computational efficiency of the developed method to a moderately sized system. The final example considers the computational model of the Cedar Avenue bridge subjected to vehicle loads and demonstrates the gains in computational efficiency attained by the proposed method for a real structural system. The final two examples also highlight the fact that the proposed method has significant advantages for larger structural models and is not limited to seismic problems only. The timing studies of the first three examples were performed on a Dell 490 Precision workstation, with a processor speed of 3.0 GHz and 8 GB of RAM while that of the fourth example was performed on a machine equipped with two 2.40 GHz Intel Xeon E5530 quad-core CPUs

with 16 GB of RAM. Matlab®, version 7.8.0.347 (R2009a) was used to implement the proposed method.

5.3.1 Example 1: Error Convergence of the Proposed Method

A base isolated 10-story shear-beam building is considered as shown in Figure 5.2. The equations of motion of the nominal building model can be written in configuration space as

$$m_b\ddot{u}_b + d_b\dot{u}_b + k_b u_b = -m_b\ddot{u}_g + d\dot{x}_1 + kx_1 \tag{5.25}$$

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{D}\dot{\mathbf{x}} + \mathbf{K}\mathbf{x} = -\mathbf{M}\mathbf{1}(\ddot{u}_b + \ddot{u}_g) \tag{5.26}$$

where d_b and k_b are the isolation parameters, u_b is the displacement of the base relative to the ground displacement, $\mathbf{x}$ is the vector of floor displacements relative to the base, $\ddot{u}_g$ is the ground acceleration, and $\mathbf{1}$ is a vector with each element equal to unity. The parameters of the nominal base isolated building were selected so the building possessed a fundamental mode with frequency 0.40 Hz and a damping ratio of 1.26%.

In the modified system, the base-isolator damping is assumed to be increased by Δd; the equations of motion for the modified base can thus be written in configuration space as

$$m_b\ddot{u}_b + (d_b + \Delta d)\dot{u}_b + k_b u_b = -m_b\ddot{u}_g + d\dot{x}_1 + kx_1 \tag{5.27}$$

while the superstructure equations of motion remain the same as in (5.26). The modified damping resulted in a building with a fundamental frequency of 0.41 Hz and 28.9% damping. One can recast the equations of motion in state-space form by defining a 22×1 state-vector, $\underline{\mathbf{v}}$, as

$$\underline{\mathbf{v}} = [u_b, x_1, \ldots, x_{10}, \dot{u}_b, \dot{x}_1, \ldots, \dot{x}_{10}]^{\mathsf{T}}. \tag{5.28}$$

The perturbation function, $\underline{\mathbf{u}}(\widehat{\underline{\mathbf{v}}})$, is linear in this case and is given by

$$\underline{\mathbf{u}}(\widehat{\underline{\mathbf{v}}}) = -\Delta d\widehat{\underline{\mathbf{v}}} = -\Delta d\dot{u}_b \tag{5.29}$$

with a scalar reduced state-vector, $\widehat{\underline{\mathbf{v}}} = \dot{u}_b$. The linear damping modification was chosen to enable an error convergence study of the proposed method as a closed-form analytical solution of the modified system response exists. Table 5.2 presents the maximum

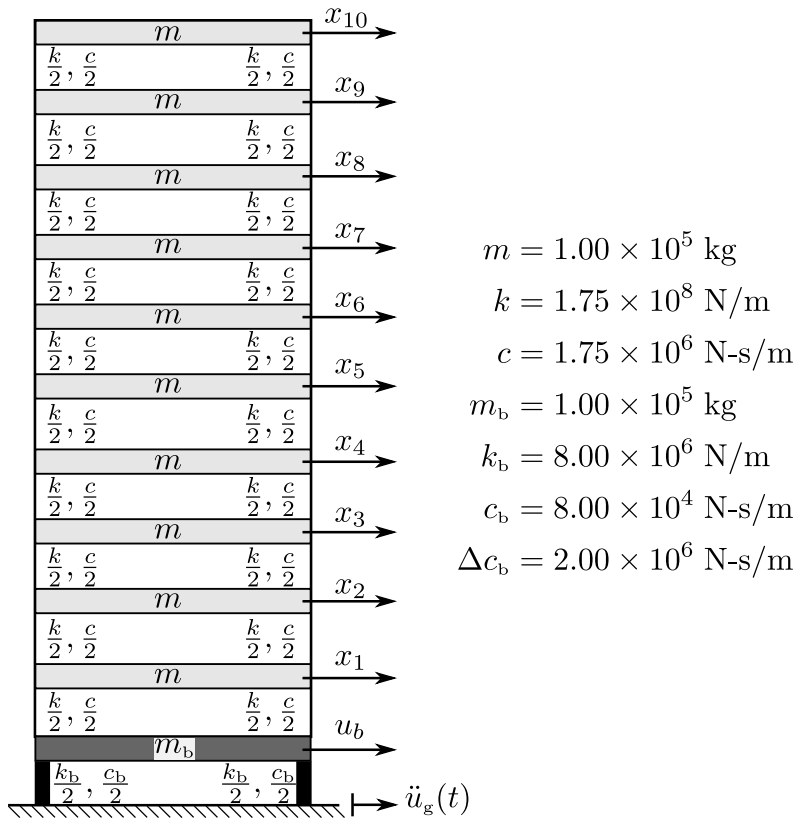


Figure 5.2: Single Multi-story Building

absolute errors among all the states as computed via the second-, the fourth- and the sixth-order quadrature rules. It should be noted that the step-size is halved in every successive row of Table 5.2, *i.e.*, $r_{\Delta t} = \Delta t_{i+1}/\Delta t_i = 0.5$. Therefore, from error convergence theory, the error ratios (e_{i+1}/e_i) are expected to converge to $r^2_{\Delta t} = 0.25$ for the second-order quadrature, $r^4_{\Delta t} = 0.0625$ for the fourth-order quadrature and $r^6_{\Delta t} = 0.015625$ for the sixth-order quadrature. It can be observed that while the second and the fourth-order quadrature rules follow the predicted theoretical trend, the sixth-order rule shows some departures from its projected theoretical trend, which can be attributed to the higher-order interpolations involved, and has been well addressed in the literature [209]. Moreover, there is not much gain in terms of accuracy in using a sixth-order rule instead of a fourth-order rule. Consequently, the fourth-order

Table 5.2: Maximum Absolute Error in all the States for Example 1

N_t	Δt	Order of Quadrature used					
		2		4		6	
		e_i	$\frac{e_{i+1}}{e_i}$	e_i	$\frac{e_{i+1}}{e_i}$	e_i	$\frac{e_{i+1}}{e_i}$
2^7	$7.87E-02$	$6.15E-02$	0.1914	$4.97E-02$	0.1256	$4.46E-02$	0.1894
2^8	$3.92E-02$	$1.18E-02$	0.2208	$6.24E-03$	0.1617	$8.45E-03$	0.0695
2^9	$1.96E-02$	$2.60E-03$	0.2422	$1.01E-03$	0.1023	$5.87E-04$	0.0129
2^{10}	$9.78E-03$	$6.30E-04$	0.2479	$1.03E-04$	0.0732	$7.56E-06$	0.0922
2^{11}	$4.89E-03$	$1.56E-04$	0.2494	$7.56E-06$	0.0661	$6.97E-07$	0.1401
2^{12}	$2.44E-03$	$3.89E-05$	0.2498	$4.99E-07$	0.0639	$9.76E-08$	0.1319
2^{13}	$1.22E-03$	$9.73E-06$	0.2499	$3.19E-08$	0.0631	$1.29E-08$	0.1282
2^{14}	$6.10E-04$	$2.43E-06$	0.2500	$2.01E-09$	0.0628	$1.65E-09$	0.1266
2^{15}	$3.05E-04$	$6.08E-07$	–	$1.26E-10$	–	$2.09E-10$	–

quadrature rule was adopted in the remaining numerical examples applying the proposed method.

This example was additionally utilized to investigate the effect of the number of partitions of the convolution space on the required computation time. A study of this kind is recommended before the application of the proposed method to unknown problems in order to determine the optimal number of partitions of the convolution space that should be utilized to maximize the computational efficiency of the method.

As the number of partitions is increased, the size of the triangles (see Figure 5.1) is decreased and more squares are introduced. This results in increasing the fraction of the convolution computation being performed using the FFT compared to that being computed directly. Thus, it is expected that the computational efficiency of the proposed method will increase with increasing number of partitions. The upper limit on the number of partitions that can be utilized depends on the number of time-points being used. The number of partitions can only be increased to a point where the smallest squares are of a size so that the overhead of the function calls becomes equal to the computation time gained by using the FFT-based convolution. Tables 5.3-5.5 demonstrate that the optimal number of partitions increase as the number of time-points is increased independent of the order of quadrature being used.

Table 5.3: Computational Time in seconds for Example 1
(2nd Order Quadrature)

N_t	RK	Number of Partitions							
		2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
2^{11}	0.18	0.41	0.41	0.39	0.39	0.39	0.39	0.39	0.40
2^{12}	0.30	0.56	0.54	0.52	0.52	0.51	0.52	0.51	0.52
2^{13}	0.58	1.20	1.06	1.01	0.99	0.96	0.96	0.96	0.97
2^{14}	1.15	2.94	2.46	2.21	2.10	2.04	2.02	2.00	2.00
2^{15}	2.31	8.29	6.35	5.37	4.88	4.66	4.55	4.49	4.51

Table 5.4: Computational Time in seconds for Example 1
(4th Order Quadrature)

N_t	RK	Number of Partitions							
		2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
2^{11}	0.34	0.53	0.53	0.52	0.52	0.52	0.52	0.52	0.53
2^{12}	0.67	0.88	0.86	0.85	0.84	0.83	0.83	0.84	0.85
2^{13}	1.33	1.95	1.84	1.77	1.74	1.73	1.73	1.72	1.73
2^{14}	2.65	4.50	4.02	3.78	3.66	3.60	3.58	3.56	3.57
2^{15}	5.30	11.28	9.34	8.41	7.88	7.66	7.57	7.48	7.49

Table 5.5: Computational Time in seconds for Example 1
(6th Order Quadrature)

N_t	RK	Number of Partitions							
		2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
2^{11}	0.62	0.76	0.75	0.75	0.75	0.75	0.75	0.75	0.76
2^{12}	1.21	1.50	1.47	1.45	1.45	1.44	1.45	1.45	1.45
2^{13}	2.44	3.15	3.02	2.96	2.94	2.91	2.92	2.91	2.92
2^{14}	4.91	6.77	6.31	6.04	5.93	5.87	5.84	5.83	5.83
2^{15}	9.79	15.68	13.72	12.77	12.31	12.06	11.93	11.89	11.87

5.3.2 Example 2: Coupled Multi-Story Building Model

Two identical, coupled shear beam buildings (same as in the previous example) are considered in this example. In the nominal system, the buildings are entirely linear

and there is no structural connection between them. If superscripts l and r denote the responses of the left and the right buildings respectively, then using a similar nomenclature as in the previous example, the equations of motion of the nominal linear system can be written as

$$m_b\ddot{u}_b^l + d_b\dot{u}_b^l + k_b u_b^l = -m_b\ddot{u}_g + d\dot{x}_1^l + kx_1^l \tag{5.30}$$

$$\mathbf{M}\ddot{\mathbf{x}}^l + \mathbf{D}\dot{\mathbf{x}}^l + \mathbf{K}\mathbf{x}^l = -\mathbf{M}\mathbf{1}(\ddot{u}_b^l + \ddot{u}_g) \tag{5.31}$$

$$m_b\ddot{u}_b^r + d_b\dot{u}_b^r + k_b u_b^r = -m_b\ddot{u}_g + d\dot{x}_1^r + kx_1^r \tag{5.32}$$

$$\mathbf{M}\ddot{\mathbf{x}}^r + \mathbf{D}\dot{\mathbf{x}}^r + \mathbf{K}\mathbf{x}^r = -\mathbf{M}\mathbf{1}(\ddot{u}_b^r + \ddot{u}_g). \tag{5.33}$$

In the modified system, the left building is assumed to be the same as that in the previous example, but with an added hysteretic nonlinearity at its base, while the right building is assumed to have an added cubic nonlinearity in its base stiffness. Also, in the modified system, the roofs of the two buildings are connected by a linear spring as shown in Figure 5.3. The superstructure equations of motion are the same as in

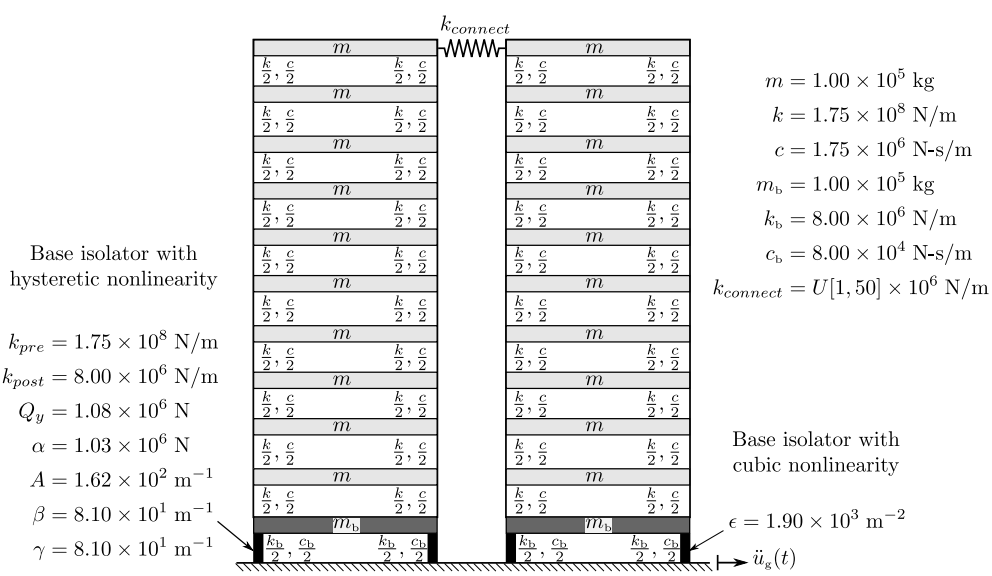


Figure 5.3: Coupled Multi-storied Building

(5.30)-(5.33) except for

$$
\begin{aligned}
m_b\ddot{u}_b^l + d_b\dot{u}_b^l + k_{post}u_b^l + \alpha z &= -m_b\ddot{u}_g + d\dot{x}_1^l + kx_1^l \\
\dot{z} &= A\dot{u}_b^l - \beta\dot{u}_b^l|z| - \gamma z|\dot{u}_b^l| \\
m_b\ddot{u}_b^r + d_b\dot{u}_b^r + k_bu_b^r + \epsilon k_b{u_b^r}^3 &= -m_b\ddot{u}_g + d\dot{x}_1^r + kx_1^r \\
F_c &= k_c(x_{10}^l - x_{10}^r)
\end{aligned}
\tag{5.34}
$$

where Q_{y} is the total force at yield, $\alpha = Q_{\mathrm{y}}[1-(k_{\mathrm{post}}/k_{\mathrm{pre}})]$ is the peak of the non-elastic force, k_{pre} and k_{post} are the pre-yield and post-yield tangential stiffnesses, respectively, and $A = 2\beta = 2\gamma = k_{\mathrm{pre}}/Q_{\mathrm{y}}$ (which makes z stay in $[-1, 1]$ and makes loading and unloading stiffnesses identical). Here, the yield force Q_{y} is chosen to be 10% of the building (superstructure plus base) weight, which is in the range typically recommended for medium-to-strong earthquakes which also gives significant yielding. ϵ denotes the strength of the cubic nonlinearity, k_c is the stiffness of the connecting spring, and F_c is the coupling force which is added with appropriate sign to the superstructure equations. Numerical values of all relevant parameters of the coupled buildings are also given in Figure 5.3. One can again recast the equations of motion in state-space form now by defining a 45×1 state-vector, $\underline{\mathbf{v}}$,

$$
\underline{\mathbf{v}} = \left[u_b^l, u_b^r, x_1^l, \ldots, x_{10}^l, x_1^r, \ldots, x_{10}^r, \dot{u}_b^l, \dot{u}_b^r, \dot{x}_1^l, \ldots, \dot{x}_{10}^l, \dot{x}_1^r, \ldots, \dot{x}^r{}_{10}, z\right]^{\mathsf{T}}. \tag{5.35}
$$

The desired outputs can be obtained via (5.14); two cases are considered: (i) three selected outputs (base displacement of second building, base velocity of first building, and relative roof velocity) are computed, with $\mathbf{C} = [\mathbf{e}_2, \mathbf{e}_{23}, \mathbf{e}_{12} - \mathbf{e}_{22}]^{\mathsf{T}}$, and (ii) all states are computed, with $\mathbf{C} = \mathbf{I}_{45\times45}$. Here, $\mathbf{e}_i$ represents a vector of length 45 having all the elements equal to zero and the ith element equal to unity, and $\mathbf{I}$ is the identity matrix.

The nonlinear vector function, $\underline{\mathbf{u}}(\widehat{\underline{\mathbf{v}}})$ takes the form:

$$
\underline{\mathbf{u}}(\widehat{\underline{\mathbf{v}}}) = -\begin{bmatrix} A\dot{u}_b^l - \beta\dot{u}_b^l|z| - \gamma z|\dot{u}_b^l| \\ \epsilon k_b{u_b^r}^3 \\ k_c(x_{10}^l - x_{10}^r) \end{bmatrix} \tag{5.36}
$$

with a 4×1 partial state vector, $\widehat{\underline{\mathbf{v}}}$, defined as

$$
\widehat{\underline{\mathbf{v}}} = \left[\dot{u}_b^l, z, u_b^r, x_{10}^l - x_{10}^r\right]^{\mathsf{T}}. \tag{5.37}
$$

The stiffness of the spring connecting the two buildings was assumed to be uncertain and modeled by a uniform random variable. A timing study was performed for one sample realization to select the number of partitions in the convolution space, which was found to be $2^4 = 16$ and a fourth-order quadrature rule was employed. 1000 samples of the system's response were computed with 2^{12} time-points for a duration of 25 seconds. Table 5.6 presents the maximum error among all the samples for the specified DOFs along with the computational time taken by the proposed method and the projected computational time that would be taken by Matlab's `ode45` function to generate 1000 samples of the system's response.

Table 5.6: Computational Times and Errors for Example 2

		Selected DOFs		**All DOFs**	
		`ode45`	Proposed	`ode45`	Proposed
CPU Time	Nominal solve†		0.08 s		0.07 s
	Impulse response†		0.10 s		0.09 s
	First $j0$ points‡		3.87 s		4.28 s
	NVIE solve‡		612.10 s		612.69 s
	FFT convolution‡		0.00 s		224.41 s
	Total‡	30 min	616.15 s ($\approx$ 10 min)	30 min	841.54 s ($\approx$ 14 min)
Maximum absolute error		-	$8.2927E-05$	-	$8.4797E-05$ ($\mathbf{x}$) $5.3411E-03$ ($\dot{\mathbf{x}}$)
†One time computation; ‡$1,000$ runs					

The model considered herein has strong nonlinearities in the form of hysteretic stiffness behavior. Moreover, it has a fairly high percentage of nonlinear and uncertain DOFs ($\approx 14\%$). The numerical stiffness of the discretized equations of motion of the modified system with respect to the nominal linear system can be scrutinized by looking at the corresponding computational times. The nominal linear system takes only about a tenth ($\approx 0.05s$) of the computational time required to solve the nonlinear system. When $1,000$ samples of the nonlinear system are generated, the proposed algorithm achieves a computational efficiency of about 3 times over the conventional approach. This gain is significant considering the relatively small size of the system, large percentage of nonlinear DOFs, and non-smooth nature of the nonlinearities.

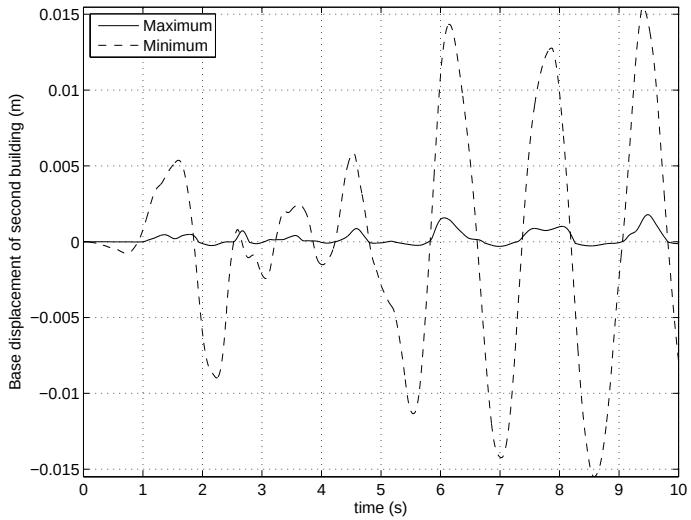


Figure 5.4: Max-min Envelope for Base Displacement of Second Building

5.3.3 Example 3: 3D Building Model with Tuned Mass Damper

A 612 DOF finite element model of a steel building subjected to wind excitation is considered with a tuned mass damper (TMD) on the roof as shown in Figure 5.5. Without the TMD, the structure has a 1.42 Hz fundamental natural frequency and 3% modal damping; the TMD is tuned to 1.38 Hz which gives the lowest peaks in the transfer function from wind excitation to roof displacement; the resulting combined system has a fundamental frequency of 1.30 Hz. The building is subjected to wind excitation in one direction (oriented along the TMD motion in Figure 5.5), modeled as a narrow-band filtered Gaussian white noise process centered near the fundamental mode of the structure without the TMD and shaped vertically as a power law function of height. The damping of the TMD has a nonlinear viscous term $c \cdot \mathrm{sgn}(v)|v|^{\alpha}$ as well as a linear viscous term; here, $c = 1$ is a scaling coefficient, v is the velocity across the damper (the velocity of the tuned mass with respect to the roof), and α is a uniform random variable on [0.5,1.5].

The state-vector is chosen to be

$$\underline{\mathbf{v}} = [x_1, \ldots, x_{612}, u_{TMD}, \dot{x}_1, \ldots, \dot{x}_{612}, \dot{u}_{TMD}]^{\mathsf{T}} \tag{5.38}$$

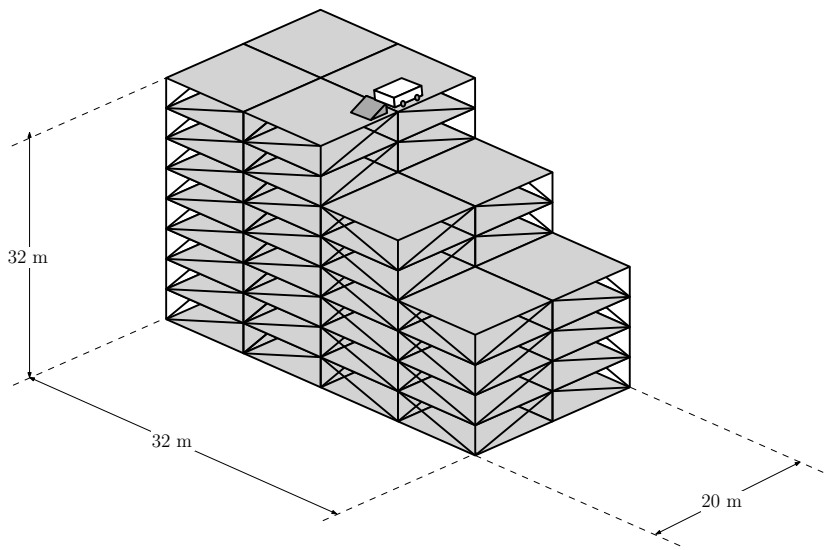


Figure 5.5: 3D Building Model with Tuned Mass Damper

and is of size 1226×1. The desired outputs are obtained via (5.14); two cases are considered: (i) three selected outputs (base displacement, roof displacement, and relative roof velocity) are computed, with $\mathbf{C} = [\mathbf{e}_1, \mathbf{e}_{565}, \mathbf{e}_{1226} - \mathbf{e}_{1178}]^{\mathsf{T}}$, and (ii) all states are computed, with $\mathbf{C} = \mathbf{I}_{1226\times 1226}$. Here, $\mathbf{e}_i$ represents a vector of length 1226 having all the elements equal to zero and the ith element equal to unity, and $\mathbf{I}$ is the identity matrix. The nonlinear function is

$$\underline{\mathbf{u}}(\underline{\widehat{\mathbf{v}}}) = c(\dot{u}_{roof} - \dot{u}_{TMD})^{\alpha} \tag{5.39}$$

with $\dot{u}_{roof} = \dot{x}_{1178}$ and the reduced state-vector, $\underline{\widehat{\mathbf{v}}}$, given by

$$\underline{\widehat{\mathbf{v}}} = [\dot{x}_{1178}, \dot{u}_{TMD}]^{\mathsf{T}} . \tag{5.40}$$

$1,000$ samples of the response were computed with 2^{12} time-points for a duration of 10 seconds. Table 5.7 presents the maximum error among all the samples for the specified DOFs along with the computational time taken by the proposed method and the projected computational time that will be taken by Matlab's `ode45` function to generate $1,000$ samples of the system's response.

It can be clearly seen that the method is much more powerful for a larger system (such as the one in this example) with a small percentage of model uncertainty. The

Table 5.7: Computational Times and Errors for Example 3

		Selected DOFs		**All DOFs**	
		`ode45`	Proposed	`ode45`	Proposed
CPU Time	Nominal solve†		30.78 s		30.78 s
	Impulse response†		34.01 s		34.01 s
	First $j0$ points‡		65.05 s		65.05 s
	NVIE solve‡		120.50 s		120.50 s
	FFT convolution‡		0.00 s		1296.92 s
	Total‡	116 hours	250.34 s ($\approx$ 4 min)	116 hours	1547.26 s ($\approx$ 26 min)
e_{max} in base displacement		-	$4.8425E-05$	-	$4.8425E-05$
e_{max} in roof displacement		-	$7.9328E-04$	-	$7.9328E-04$
e_{max} in rel. roof velocity		-	$9.5672E-02$	-	$9.5672E-02$
†One time computation; ‡$1,000$ runs					

computational efficiency obtained when only 3 of the 612 DOFs are computed is ≈ 1700 times; it is ≈ 250 times when all the DOFs are computed. It should be noted that this is still a moderately sized system (with 612 DOFs). In real applications, where system sizes are of the order of thousands of millions of DOFs with the perturbations (uncertainties/nonlinearities) present in a very small subset of the system DOFs, the proposed method promises to be even more computationally efficient.

5.3.4 Example 4: Cedar Avenue Bridge Model

The final example considers the computational model of an actual bridge, the Cedar Avenue bridge, in order to demonstrate the computational efficiency of the proposed method when applied to a real life system. The bridge is located in the metro area of the Twin Cities in Minnesota. Thompson and Schultz [204] created a finite element model of the Cedar Avenue Bridge in SAP2000 [205] using 3D frame and shell elements. Figure 3.8 shows a photograph and the finite element model of the Cedar Avenue bridge.

The bridge model is equipped with a scissor jack apparatus as shown in Figure 3.9(a) to provide counter moments in order to reduce the moment range experienced by certain joints. Gastineau *et al.* [206] developed the finite element model of the modified bridge consisting of $17,687$ degrees-of-freedom as shown in Figure 3.9(b). The modification

apparatus is connected across four joints, thereby causing a rank-8 perturbation in the system's stiffness matrix. The stiffness properties of the modification apparatus are assumed to be uncertain, modeled using a uniform distribution within a 4% range of the corresponding nominal values. The bridge model is subjected to a moving truck load with the specifications of the truck given by the AASHTO standards for fatigue loading [210]. The standard truck specified in the AASHTO standards is shown in Figure 5.6.

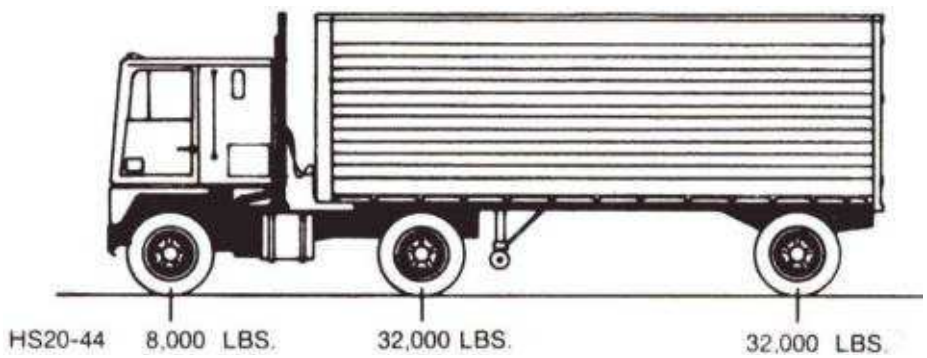


Figure 5.6: AASHTO Standard Truck

The response of the nominal system has been computed using SAP2000 [205] on a machine equipped with a 3.4 GHz Intel core i7 CPU with 16 GB of RAM. SAP2000 utilizes the Hilber-Hughes-Taylor (HHT) method (also known as the alpha-method) [211] for time integration. Since the HHT method is second-order accurate, a second-order quadrature has been utilized when computing the system's response via the proposed method. A 5 second time history has been computed with 2^{10} time-points; the convolution space has been sub-divided into 16 partitions. Table 5.8 shows the required computational times and gains in the computational efficiency for the computation of $1,000$ and $10,000$ sample realizations of the dynamic response of the modification apparatus. The proposed method achieves $\mathcal{O}\left(10^3\right)$ gain in computational efficiency when $10,000$ sample realizations are computed. The computational gains observed in this example are not as pronounced as in the previous example even though the size of the computational model is larger. This behavior is most likely caused by:

- the relatively full stiffness matrix of the bridge model (99.95% non-zero entries) which increases the required number of computational operations

Table 5.8: Computational Times and Efficiency Gain Ratios for Example 4

	1,000 samples	**10,000 samples**
Brute-force method CPU time (t_b)	6.2 days	2.1 months
Proposed method CPU time (t_p)	19 minutes	1.5 hours
Computational efficiency (t_b/t_p)	470	1008

- the use of a superior CPU (3.4 GHz) to compute the solution via the brute-force method (compared to 2.4 GHz CPU used to compute the solution via the proposed method)

5.4 Conclusions

This chapter presented a method for the rapid uncertainty quantification of the time domain response of the dynamical systems with local nonlinearities and/or uncertainties in the stiffness and/or damping properties of the system. The proposed method first computed the response of a related linear system and then rapidly computed the response of an ensemble of locally modified systems as the sum of this nominal linear response and a modification term. This modification term was determined through the solution of a nonlinear Volterra integral equation written in a non-standard form. A fast solution algorithm employing Newton-Gregory quadrature and a recursive convolution methodology for the governing nonlinear Volterra integral equation was presented.

Four numerical examples were presented to demonstrate the error behavior and the computational efficiency of the proposed method. The first example demonstrated the error behavior of the proposed method along with the effect of the number of sub-divisions of the convolution space on the computational time required by the proposed method.

The next two examples presented Monte Carlo simulation-based uncertainty analyses of two representative systems with both local uncertainty and local nonlinearity. Gains in computational efficiency of as high as 1700 times were observed when $1,000$ samples of 3 out of 612 states were computed for the three-dimensional building model. Even when all the states were computed for the model, the gain in efficiency was still 250 times. The nonlinearity in this example was relatively smooth and the size of the system was

fairly large. For a smaller system, with 22 DOFs and with a non-smooth nonlinearity, a gain in computational efficiency of 3 times was observed for the simulation of $1,000$ samples.

The final example considered a large computational model of a real structural system and the proposed method demonstrated gains in computational efficiency of $\mathcal{O}\left(10^2\right) - \mathcal{O}\left(10^3\right)$. Thus, the proposed method showed great promise in its applicability to the time domain analysis of dynamical systems with localized nonlinearities and uncertainties, and promises to be extremely useful allowing one to obtain tens to thousands more samples than the brute-force Monte Carlo simulation for a fixed computational time.

Chapter 6

Parallel Computing for Uncertainty Quantification

The principal aim of this chapter is to explore the use of a desktop computer equipped with a GPU to perform uncertainty analysis of computational models in an efficient manner. The computational performance of the GPU implementations developed in this chapter will be compared to equivalent implementations on single- and multi-core CPUs. The optimality of these implementations will be discussed as needed. It should be noted that the optimality of sequential and/or parallel CPU-based implementations is not the focus of this chapter. Therefore, most of the ensuing discussion will be concerned with GPU implementations. However, certain performance metrics related to CPU-based parallel implementations will be discussed.

Figure 6.1 shows a conceptual time-line (order of execution) of a serially-executed sampling-based uncertainty quantification procedure. $\mathcal{S}_i$ represents the computation of

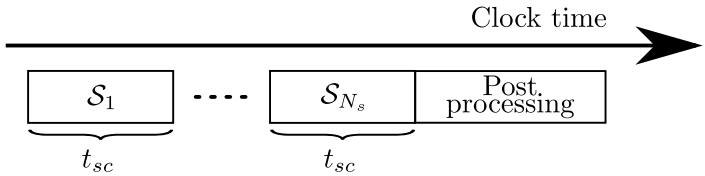


Figure 6.1: Serial Implementation of Sampling-based UQ

the response of the ith sample of the underlying system. t_{sc} represents the computational time required to compute one sample of the system's response using one computing process. Figure 6.1 will form the basis of the discussion of the parallelization strategies in subsequent sections. Since the developments in this chapter are concerned with only the parallelization of the sampling process and the computation of the system's response, the computational time consumed by post-processing operations (*e.g.* computation of relevant statistics from response samples) will not been considered. However, as will be discussed later, it is possible to reduce the post-processing time as well. It is evident that the total computation time required to generate all of the samples of the system's response via a serial implementation will be

$$t_{total}^{(s)} = N_s t_{sc}. \tag{6.1}$$

In the next section, parallelization strategies that can be employed for uncertainty quantification of dynamical and mechanical computational models will be discussed. Next, the advantages of using GPUs for general purpose computing will be elaborated upon along with the inherent challenges encountered in their application. Finally, GPU implementations of five analyses procedures pertaining to uncertainty quantification of dynamical and mechanical systems will be discussed. In addition, a GPU implementation of the efficient time domain analysis method proposed in Chapter 5 will also be presented.

6.1 Parallelization Strategies

Chapter 1 introduced three strategies that can be employed to parallelize uncertainty quantification procedures. These strategies are illustrated in Figure 6.2. The subsequent

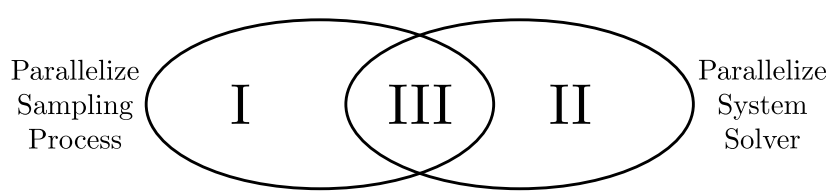


Figure 6.2: Types of Parallelization Strategies

subsections discuss each of these three strategies in detail.

6.1.1 Type I: Parallelize the Sampling Process

Parallelization of the sampling process is relatively straightforward as discussed in Section 1.3. Figure 6.3 shows a conceptual implementation of such a strategy. Each computing process, p_i, computes N_i samples of the system's output such that

$$\sum_{i=0}^{N_{proc}-1} N_i = N_s \tag{6.2}$$

where N_{proc} is the total number of parallel computing processes involved and N_s is the total number of samples required. The subscripts on $\mathbf{y}$ in Figure 6.3 denote sample numbers. The computing processes involved may be as intensive as a full processor

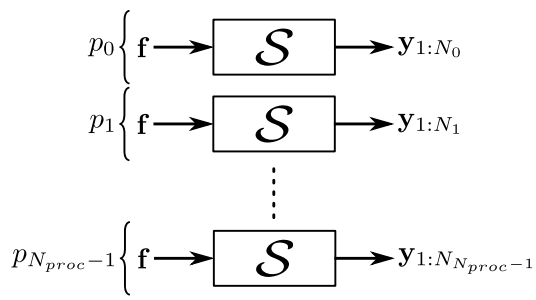


Figure 6.3: Type I Parallelization Strategy

or as light as a single computational thread depending upon the size of the system, $\mathcal{S}$, involved. Also, the number of samples computed by each process, N_i, can be changed in accordance with the number of available processes resulting in greater control over the granularity of the implementations. Further, since each process computes independent samples of the system's response, inter-process communication is minimal.

These two characteristics, flexible levels of granularity and minimal inter-process communication, render such a parallelization strategy a prime candidate to be implemented on a GPU due to their single-instruction multiple-thread (SIMT) design philosophy as discussed in Section 1.3. Figure 6.4(a) shows the conceptual time-line associated with a Type I parallelization strategy. t_p represents the computational

time required by each process to compute one sample of the system's response. If the computational processes utilized herein possess similar capabilities as the serial process utilized in Figure 6.1, it is evident that $t_p \approx t_{sc}$. Thus, the total computational time required when employing this strategy is given by

$$t_{total}^{(\mathrm{I})} = \frac{N_s}{N_{proc}} t_p \approx \frac{N_s}{N_{proc}} t_{sc} \tag{6.3}$$

where N_s/N_{proc} is the average number of samples computed by each process. Also, the inter-process communication time has been neglected in (6.3) due to reasons discussed earlier. From (6.1) and (6.3), the expected gain in computational efficiency obtained

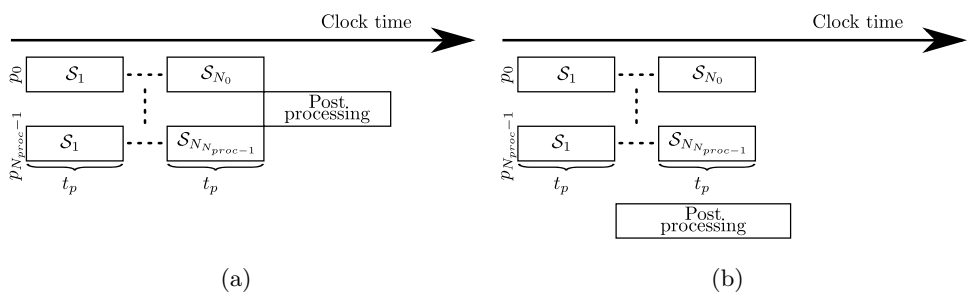


Figure 6.4: Implementation of Type I Parallelization Strategy

can be found to be

$$S_{\mathrm{I}} = \frac{t_{total}^{(s)}}{t_{total}^{(\mathrm{I})}} \approx N_{proc}. \tag{6.4}$$

Additional application-level computational efficiency may be derived by initiating the post-processing as soon as the first sample of the system's response has been computed by each process, as shown by Figure 6.4(b).

6.1.2 Type II: Parallelize the System Solver

The second strategy aims at parallelizing the system solver itself by utilizing the dwarfs of parallel computing discussed earlier and shown in Figure 1.3. In this strategy, the available computing processes work simultaneously to compute each sample of the response of a given computational model. Figure 6.5 shows a possible dependency flowchart of system solvers on the dwarfs of parallel computing. The computing processes may be as intense as a cluster of processors or as light as an agglomeration of

parallel threads depending upon the size of the computational model. The inter-process communication cost associated with this strategy is oftentimes non-trivial and therefore, such a strategy might be beneficial for relatively large-scale computational models as the computational cost is significantly greater than the communication cost. Moreover, the granularity of such implementations depends greatly on the structure of the system matrices and the algorithms involved.

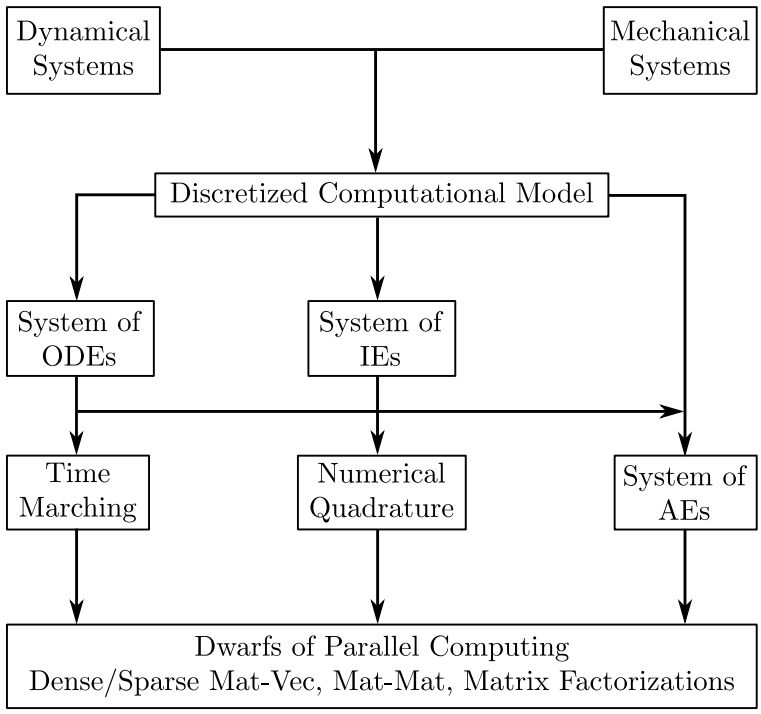


Figure 6.5: Type II Parallelization Strategy

Figure 6.6 shows the conceptual time-line associated with Type II parallelization strategy. t_p and t_c represent the computational and communication time required by all the processes to collectively compute one sample of the system's response, respectively. It is evident that due to the design of such a strategy $t_p < t_{sc}$. The total computational time required when employing this strategy is thus given by

$$t_{total}^{(\mathrm{II})} = N_s(t_p + t_c). \tag{6.5}$$

From (6.1) and (6.5), the expected gain in computational efficiency can be found to be

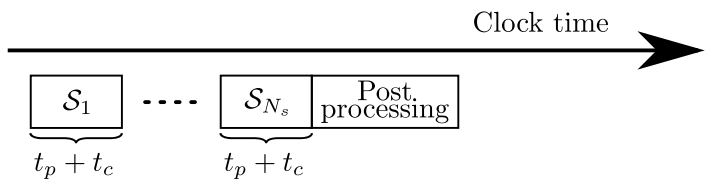


Figure 6.6: Implementation of Type II Parallelization Strategy

$$S_{\mathrm{II}} = \frac{t_{total}^{(s)}}{t_{total}^{(\mathrm{II})}} = \frac{t_{sc}}{t_p + t_c}. \tag{6.6}$$

6.1.3 Type III: Combination of Type I and Type II

The two strategies just discussed can be combined to form a third strategy that utilizes two levels of parallelism. Figure 6.7 illustrates such a strategy. Parallel processes p_i's are

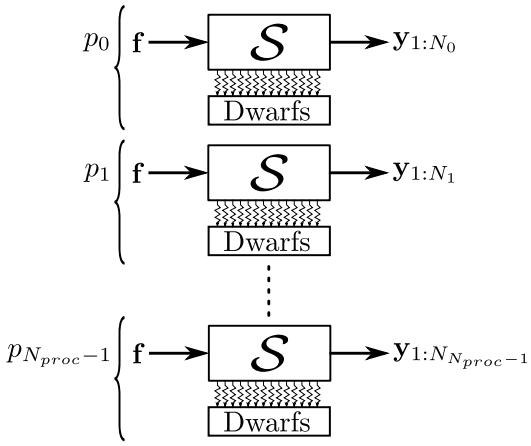


Figure 6.7: Type III Parallelization Strategy

utilized to compute N_i samples of the system's response. These processes themselves are comprised of smaller parallel processes that can employ the dwarfs of parallel computing to collectively compute each sample of the system's response. Figure 6.8 shows a conceivable two layer approach using CPUs. The first layer would be comprised of distributed computing nodes while the second layer may be formed by shared-memory threads (*e.g.* POSIX threads).

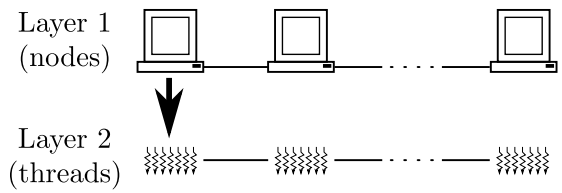


Figure 6.8: Two-level Parallelism using CPUs

Figure 6.9 shows the conceptual time-line associated with a Type III parallelization strategy. t_p and t_c again represent the computational and communication time required by the involved processes to collectively compute one sample of the system's response. Also, as in a Type II strategy, $t_p < t_{sc}$. The total computational time required when

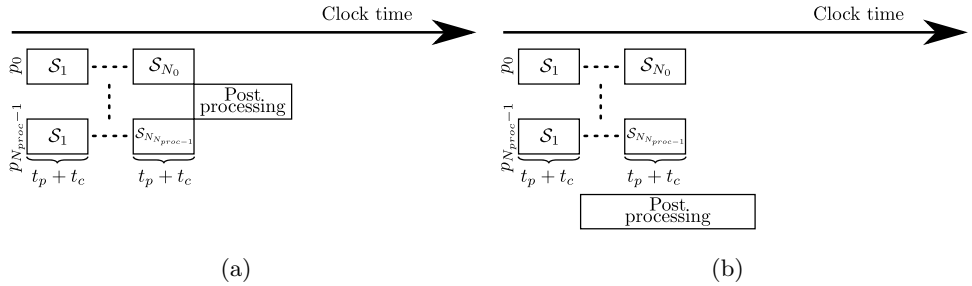


Figure 6.9: Implementation of Type III Parallelization Strategy

employing this strategy is given by

$$t_{total}^{(\mathrm{III})} = \frac{N_s}{N_{proc}}(t_p + t_c) \tag{6.7}$$

where N_s/N_{proc} is the average number of samples computed by each process as in Type I strategy. The expected gain in computational efficiency can be found to be

$$S_{\mathrm{III}} = \frac{t_{total}^{(s)}}{t_{total}^{(\mathrm{III})}} = N_{proc}\frac{t_{sc}}{t_p + t_c}. \tag{6.8}$$

6.2 GPU-based Parallel Computing

Section 1.3 introduced the notion of utilizing GPUs for uncertainty quantification procedures and the following advantages of GPGPU were highlighted:

- anticipated evolution of GPUs
- better performance of GPUs
- cost efficiency
- energy efficiency

This section will address the usability of GPUs from an algorithm-development perspective. In addition, potential challenges in the development of GPU implementations of existing algorithms will be discussed. Remarks specific to NVIDIA's Tesla C1060 GPU and NVIDIA's CUDA development environment will be provided as needed to facilitate the developments in subsequent sections. Further details of the hardware implementation of NVIDIA GPUs and CUDA programming interface can be found in the NVIDIA CUDA Programming Guide [153].

6.2.1 Advantages of GPU-based Parallel Computing

GPUs provide a scalable array of streaming multiprocessors (SMs). NVIDIA's Tesla C1060 GPU contains 30 SMs each having 8 cores, providing 240 computational cores. These SMs are designed to execute hundreds of concurrent threads employing a unique architecture, SIMT, as discussed earlier. Each SM creates, manages, schedules and executes threads in groups of 32 threads called warps. Each warp contains threads of consecutive, increasing thread IDs and executes one common instruction at a time. If threads of a warp diverge via a data-dependent conditional branch, the SM serially executes each possible branch path until all paths are complete. Although this simple control logic imposes certain constraints on a given GPU implementation, also termed a CUDA kernel, to maintain optimal execution, it also allows GPU threads to be extremely lightweight and scalable.

At the software level, GPU threads are hierarchically organized into blocks of threads and grids of blocks as shown in Figure 6.10. As such, a grid can contain a two-dimensional array of blocks, and a block can contain a three-dimensional array of threads. Such a logical organization of threads enables one to construct a one-to-one mapping between the decomposed domain of a given problem and its GPU implementation. For instance, two-dimensional blocks of threads can be utilized for

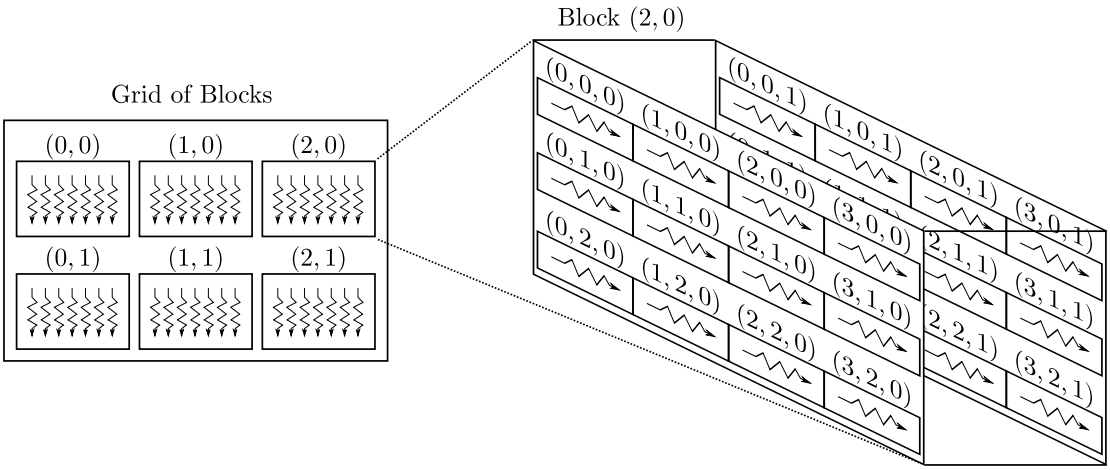


Figure 6.10: Thread Hierarchy for GPUs

algorithms that deal with two-dimensional matrices. In terms of hardware mapping, threads are assigned to SMs as blocks, *i.e.*, each SM executes individual blocks of thread one after the other. Inter-block communication is not allowed while threads within a block can communicate with each other. This feature again imposes certain constraints on a given GPU implementation but also facilitates simpler control of threads and increases the scalability of developed GPU implementations considerably.

The hardware and software features of GPUs just discussed may be utilized to conceive multiple layers of parallelism as shown in Figure 6.11. The first layer consists of distributed computing nodes while the second layer is comprised of multiple GPUs with a single computing node. The remaining three are formed on the basis of the hierarchy of threads available for GPUs. These additional layers of parallelism available when using GPUs as opposed to those available when employing solely CPU-based parallel computing (Figure 6.8) enable one to address problems of varying sizes with much more flexibility. Moreover, a single GPU can theoretically provide almost a TFLOP/s of computing power, as discussed earlier, and thus desktop machines equipped with GPUs can be utilized to efficiently solve reasonably sized problems.

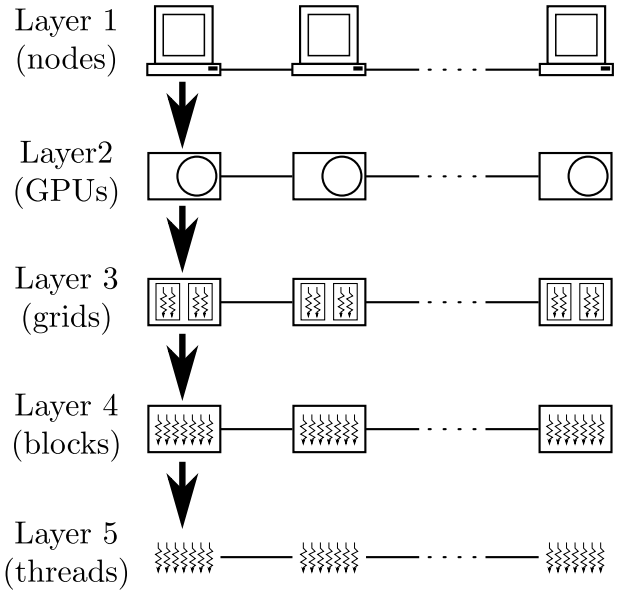


Figure 6.11: Layers of Parallelism using GPUs

6.2.2 Challenges in GPU-based Parallel Computing

Most of the challenges faced when utilizing GPUs for general purpose computing arise due to two reasons: (i) the fact that they are co-processors to CPUs and (ii) their design philosophy. Since GPUs are co-processors, as shown in Figure 6.12, all CUDA kernels must be initiated by a CPU and the relevant data must be transferred to the GPU memory before it can be processed. The bandwidth of the peripheral component interconnect (PCI) bus can become a bottleneck in a data-intensive application; therefore, data-transfer between CPU and GPU must be minimized in order to achieve better performance.

Another memory-related performance issue arises due to the relatively small size of the on-board DRAM of GPUs. For NVIDIA's Tesla C1060 GPU, 4 GB of on-board DRAM is available. The problem size that can be addressed by a particular CUDA kernel is directly affected by the available on-board memory. When a given application requires a large amount of data to be processed, the data must be processed in batches which increases the amount of data-transfer between CPU and GPU.

Yet another memory-related performance consideration results from the different

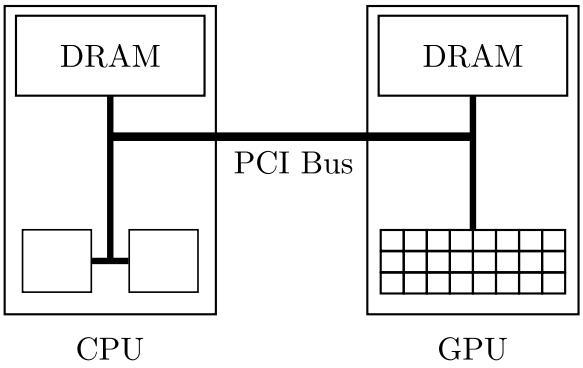


Figure 6.12: GPUs as Co-Processors of CPUs

types of memory banks available on a GPU. Usually, accessing the DRAM, also called the global memory, may require several hundred clock cycles. Therefore, repeated global memory accesses within a CUDA kernel can significantly hamper its computational performance. SMs have small amounts of on-chip memories available for use by threads that provide almost instantaneous data-access (usually less than 10 clock cycles). These on-chip memories include registers which are private to each thread and shared-memory which is shared by the threads within a block. In the latest NVIDIA GPUs, part of the shared-memory can also be utilized as a cache. A common strategy to minimize the global memory access within CUDA kernels is to process the data batch-by-batch with the size of batches being governed by the size of the on-chip memory. However, in doing so, global synchronization points must be introduced in the CUDA kernel which may render its performance suboptimal. Since the size of these on-chip memories is relatively small (16 KB of shared memory and 16K 32-bit registers on NVIDIA's Tesla C1060), care must be take while designing a CUDA kernel to ensure their optimal utilization.

The computational performance of a given GPU implementation is highly hardware-dependent due to the different data-partitioning required on different GPUs to optimally utilize their on-chip and global memories. Therefore, it is not guaranteed that a GPU implementation that performs well on a given GPU to perform equally well on another GPU, especially when the GPUs in question have been manufactured by different vendors. Several efforts are being made to address this issue through the development of additional software layers to automatically optimize a given GPU

implementation with respect to a given hardware [212–214].

Another consideration while developing GPU implementations is the increased performance of GPUs in performing single-precision (SP) computations compared to that in double-precision (DP) computations. For NVIDIA's Tesla C1060 GPU, the peak theoretical SP performance is 933 GFLOP/s while the peak theoretical DP performance is only 125 GFLOP/s. Consequently, a distinction will be made between the SP and the DP performance of the GPU implementations presented later in this chapter. The latest available GPUs from NVIDIA have considerably improved the DP performance, and this performance gap should be of lessened concern in future applications.

6.2.3 Strategies to Utilize GPUs for Uncertainty Quantification

Section 6.1 presented three strategies that can be employed to parallelize a given uncertainty quantification procedure. In order to implement the parallel algorithms developed using these strategies on a GPU, three approaches may be followed:

- Type 1: use of existing, pre-compiled, optimized libraries
- Type 2: development of new GPU implementations
- Type 3: mixed use of existing libraries with new GPU implementations as needed

The Type 1 approach is possibly the easiest to implement as it only requires adapting a given application to the format required by the algorithms available in a given library. Many optimized libraries being developed for GPUs utilize similar data-structures and algorithm organization as some of the commonly used CPU-based libraries, such as BLAS, LAPACK and FFTW. Table 6.1 presents a list of available GPU libraries that can be used with C/CUDA implementations. In addition to the libraries mentioned in Table 6.1, GPU-accelerated toolboxes have been developed for Matlab as well. Moreover, the latest versions of many of these libraries utilize both CPUs and GPUs, *i.e.*, they deploy GPUs as accelerators rather than executing the entire algorithm on them which further adds to the usability and computational efficiency of the algorithms.

While the use of these libraries ensure optimal use of GPU in most scenarios, the Type 1 approach suffers from several disadvantages. One is the need to adapt the desired application to the format required by the libraries which greatly reduces the

Table 6.1: List of GPU Libraries

Library	**Vendor**	**Comments**
CUBLAS	NVIDIA	Dense matrix algebra [157] (BLAS [215] equivalent)
CUSPARSE	NVIDIA	Sparse matrix algebra [216]
CUFFT	NVIDIA	FFT algorithms [158] (FFTW [217] equivalent)
CURAND	NVIDIA	Random number generation algorithms [218]
CUDA NPP	NVIDIA	Parallel programming primitives [219] (sort, reduction, *etc.*)
Thrust	NVIDIA	Standard template library [220]
MAGMA	MAGMA	Dense linear algebra package [160] (LAPACK [221] equivalent)
CULA Tools	EM Photonics	Dense linear algebra package [222] (LAPACK [221] equivalent)

flexibility of implementations. The unavailability of the source code of most of these libraries further reduces their flexibility as it is preferable to embed the actual GPU implementation in a given application instead of executing the same as a function call in order to maintain optimal usage of GPUs. Another drawback is that none of the libraries is yet capable to accommodate problems with sizes larger than the available GPU memory.

The above mentioned limitations may be avoided by utilizing a Type 2 approach, developing new GPU implementations. While this approach might be the most flexible and may yield best gains in computational efficiency, it requires considerable technical expertise to develop optimized GPU implementations. A more amenable approach may be a Type 3 approach where the existing libraries are utilized wherever optimal and new GPU implementations are developed when either the required algorithms are unavailable from an existing library or their use renders the entire application suboptimal.

6.3 Performance Metrics for Parallel Algorithms

The execution time of a parallel algorithm depends not only on input size but also on the number of processing elements utilized and their relative computation and interprocess communication speeds. Since only shared-memory parallelism has been utilized for the CPU-based parallel implementations herein, the communication time between processes will be ignored. Moreover, all the computational cores that execute a given parallel algorithm possess identical computational performance. For GPU-based parallel implementations, a distinction between parallel execution time and data transfer time will be made where necessary. Assuming the aforementioned conditions, the speedup of a given parallel algorithm can be defined as

$$S = \frac{t_{sc}}{t_p + t_c} \tag{6.9}$$

where t_{sc} is the sequential execution time, t_p is the computational time taken by p processors to solve the same problem and t_c is the communication or data transfer time. For GPU-based parallel implementations, t_c represents the data-transfer time between CPU and GPU. It is evident that t_c is bounded by the theoretical speed of the PCI bus as discussed earlier.

For the sequential algorithms, a measure of the theoretical FLOP count of a given algorithm in addition to its parallel execution time gives important insights into the optimality of the pertinent parallel algorithm as discussed in Section 2.7. A detailed discussion on the optimality and scalability of CPU-based parallel algorithms can be found in the literature [223]. For the examples presented later in this chapter, a multi-core CPU implementation will be assumed to be optimal in performance if the observed gain in computational efficiency approaches the number of available computational cores, *i.e.*, S_{I} for Type I strategy and S_{II} with t_c =0 for Type II strategy. Since a desktop-based machine has been utilized, Type III strategy cannot be utilized while using only CPU-based parallel computing.

For GPU-based parallel algorithms, rigorous performance metrics are not available which renders a combined study of the execution time and the observed and theoretical FLOP count of a GPU-based algorithm more important. Moreover, the fact that GPU computational cores behave significantly differently as compared to CPU computational

cores prohibits direct use of CPU-based parallel computing theory in GPU-based parallel computing.

Establishing a theoretical peak for the performance gain of a GPU implementation is highly problem dependent due to hardware constraints which govern the optimization of the GPU implementation for performance as discussed earlier. The ensuing discussion of the GPU kernel-optimization utilizes the terms memory-only performance and compute to global memory access (CGMA) ratio.

Memory-only performance is defined as the global memory throughput obtained when only memory read/write operations are performed and all the computations are omitted. This is an important metric to indicate whether the global memory accesses are coalesced. The closer the observed memory throughput is to the theoretical peak memory bandwidth (102 GB/s for NVIDIA's Tesla C1060 GPU), the better. The details of the effect of global memory access pattern (coalesced vs non-coalesced access) and shared-memory bank conflicts can be found in the literature [153].

The CGMA ratio is defined as the ratio of the total number of computations to the total number of global memory accesses. This ratio indicates whether the GPU implementation is compute- or memory-bound. When compute-bound, the reference peak performance of the GPU implementation is governed by the theoretical peak GFLOP/s of the device (933 GFLOP/s for SP, 125 GFLOP/s for DP for NVIDIA's Tesla C1060 GPU). When memory-bound, the peak theoretical performance can be computed using the peak theoretical bandwidth as

$$\text{peak theoretical performance} = \text{CGMA} \cdot \frac{\text{peak theoretical bandwidth}}{B} \tag{6.10}$$

where B is the number of bytes required to store the data ($B = 4$ for SP, $B = 8$ for DP). Thus, the theoretical peak performance of the GPU implementation presented in this study will be memory-bound if

$$\text{CGMA} < \begin{cases} 36.6 & \text{for SP} \\ 9.8 & \text{for DP} \end{cases} \tag{6.11}$$

otherwise, it will be compute-bound. In the performance evaluation of GPU implementations presented in the next section, the CUDA visual profiler has been utilized to obtain the memory-throughput while the CGMA ratio has been computed

manually. Moreover, the largest available problem-size considered has been utilized to estimate the memory-throughput and the computational performance of the algorithms.

6.4 Numerical Examples

This section discusses the computational efficiency of five uncertainty analyses procedures that utilize GPUs. These five procedures have been implemented entirely on a GPU, *i.e.*, they perform all the computations on the GPU. Their performance is demonstrated by the first five examples. The final example demonstrates hybrid implementation of the efficient time domain analysis method developed in Section 5.2. The analysis procedure presented in Algorithm 6 is implemented to utilize both CPU and GPU.

The computational times required by GPU implementations are compared to that of corresponding single- and multi-core CPU implementations for each method. One NVIDIA Tesla C1060 GPU and two Intel Xeon E5530 quad-core CPUs have been utilized to perform all the timing studies. Moreover, timing comparisons have been performed for both SP and DP computations for all the examples. Since there is a negligible performance gap in SP and DP computations performed on a CPU, such a distinction has not been made for the multi-core CPU computational gains.

The gains in computational efficiency, utilized in the discussion of the subsequent examples, are defined as follows:

$$S_{MC} = \frac{t_{sc}}{t_{mc}} \tag{6.12}$$

$$S_{GS} = \frac{t_{sc}}{t_{gp}} \tag{6.13}$$

$$S_{GM} = \frac{t_{mc}}{t_{gp}} \tag{6.14}$$

where t_{sc}, t_{mc} and t_{gp} are the computational times required by the single-core CPU, multi-core CPU, and GPU implementations of a given procedure respectively. S_{MC} represents the gain in computational efficiency obtained by the multi-core CPU implementation compared to the single-core CPU implementation, and S_{GS} and S_{GM} are the gains in computational efficiency obtained by the GPU implementation compared to the single- and multi-core CPU implementations, respectively.

6.4.1 Example 1: Static Analysis of a Linear System

The first example considers a cantilevered Eüler-Bernoulli beam as shown in Figure 6.13. The beam is subjected to a random, static tip load. A finite element discretization with cubic shape functions has been utilized to obtain the stiffness matrix, $\mathbf{K}$, and the load vector, $\mathbf{f}$. The static response of such a system is given as

$$\mathbf{x} = \mathbf{K}^{-1}\mathbf{f}. \tag{6.15}$$

The static deflection of the beam is computed using (6.15) due to 1000 different samples of the load. The `sgetrf` and `dgetrf` routines have been utilized from the Intel's MKL [224] library, and their inherent parallelism has been utilized for the multi-core CPU implementation. The corresponding routines from the MAGMA [160] library have been utilized for the GPU implementation; both have been assumed to be optimal in performance based on the evidence available in their respective documentations. This example utilizes a Type III parallelization strategy and a Type 1 implementation approach.

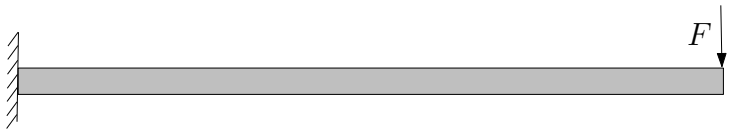


Figure 6.13: Cantilever Beam Model

Table 6.2 shows the gains in computational efficiency obtained for this example. The GPU implementation shows gains in computational efficiency of approximately 200 times compared to the single-core CPU implementation and approximately 30 times compared to the multi-core CPU implementation for both SP and DP computations for the largest considered problem size.

6.4.2 Example 2: Spectral Analysis of a Linear System

The second example considers the same beam model as in Example 1 but is now subjected to a random dynamic load, governed by a first-order autoregressive, stationary random process which is prescribed in the form of its PSD. The analysis is performed in

Table 6.2: Gains in Computational Efficiency for Example 1

# DOF	S_{MC}	S_{GS}		S_{GM}	
		SP	DP	SP	DP
128	1.00	1.95	2.00	1.95	2.00
512	1.56	3.67	6.25	2.35	4.01
1024	3.88	2.25	2.38	0.58	0.61
2048	6.31	4.04	3.56	0.64	0.56
4096	6.06	5.97	4.19	0.99	0.69
8192	6.46	32.10	27.65	4.97	4.28
16384	7.31	213.85	205.33	29.25	28.09

the frequency domain, and the PSD of the response of all the states is directly computed using (2.52). Since the input is present in only one degree-of-freedom of the system, (2.52) can be computed by employing a rank-1 update algorithm followed by scaling with a complex scalar. The rank-1 update has been performed using the `cherk` and `zherk` BLAS routines from the Intel's MKL library in the CPU implementation. The scaling with the complex scalar has been performed using a vectorized loop obtained by aggressive optimization of the code via the compiler. Inherent parallelism of the MKL BLAS routines has been exploited to optimize the multi-core CPU performance. On the GPU, the rank-1 update is performed using the corresponding BLAS routines from the CUBLAS [157] library. A new GPU kernel has been developed for the scaling. Thus, this example utilizes a Type II parallelization strategy and a Type 3 implementation approach.

The gains in the computational efficiency obtained for this example are shown in Table 6.3 as a function of system size. The multi-core CPU implementation doesn't show much gain in computational efficiency because of the relatively small size of the matrices involved. The GPU implementation, however, shows about 70 times better performance for SP and about 20 times better performance for DP computations for the largest considered problem size.

About 98% of the computational time of the GPU implementation is consumed by the BLAS routines which have been assumed to be already optimized for performance. The remaining portion, which scales the involved matrix-matrix product to obtain the spectral response of the system, is memory-bound, with a CGMA ratio of 2.25 and has

Table 6.3: Gains in Computational Efficiency for Example 2

# DOF	S_{MC}	S_{GS}		S_{GM}	
		SP	DP	SP	DP
100	0.32	0.91	0.48	1.02	1.50
500	0.97	14.71	6.09	15.68	6.30
1000	1.31	34.14	14.16	35.27	10.80
5000	1.32	68.17	27.05	66.76	20.42
7500	1.32	71.78	28.47	67.97	21.49
10000	1.32	73.82	29.29	71.17	22.11

been assumed optimal in performance because it shows a computational performance of 50.67 GFLOP/s in SP and 23.82 GFLOP/s in DP compared to the theoretically predicted performances of 57.38 GFLOP/s and 28.69 GFLOP/s respectively. This kernel utilizes n^2 threads, n being the number of degrees-of-freedom of the system.

6.4.3 Example 3: Time Domain Analysis of a Linear System

The third example considers the same beam model as in the previous two examples but it is now subjected to a time-dependent random load, governed by a first order autoregressive, stationary random process. The equations of motion for this example are written as in (2.2) and the response is computed using the convolution integral as given by (2.30). 100 samples of the response of all the states were computed.

The convolution was performed using the FFT which requires three steps: (i) computing the FFT of the two vectors, (ii) computing the element-by-element product of the transformed vectors, and (iii) computing the inverse FFT of the product. The FFTW [217] and the CUFFT [158] libraries have been utilized to perform the FFT and the inverse FFT on CPU and GPU, respectively. The built-in shared-memory parallelism of the FFTW library has been utilized along with a pthread implementation of the element-by-element multiplication to obtain the multi-core CPU implementation. A custom kernel has been implemented to perform the element-by-element product on the GPU. A Type III parallelization strategy and a Type 3 implementation approach has been employed in this example.

The gain in computational efficiency obtained using the multi-core CPU

implementation approaches 8 and hence, is considered to be optimal. Table 6.4 shows the gains in computational efficiency obtained by the GPU implementation for different problem sizes. For the largest considered problem size, the GPU implementation shows a gain in computational efficiency of approximately 1200 and 180 times for SP and DP computations compared to the single-core CPU implementation and about 880 and 120 times compared to the multi-core CPU implementation for SP and DP computations, respectively.

Table 6.4: Gains in Computational Efficiency for Example 3

# DOF	S_{MC}	S_{GS}		S_{GM}	
		SP	DP	SP	DP
$N_t = 1024$					
60	6.23	245.44	132.54	45.11	21.26
124	6.84	233.62	145.32	37.49	21.26
252	7.33	294.83	151.90	44.01	20.73
$N_t = 8192$					
64	7.36	337.54	384.71	54.52	52.28
128	7.42	692.61	806.69	102.65	108.69
256	7.40	1206.95	881.04	181.84	119.05

In the GPU implementation, the FFT operations consume only about 20% of the computational time. Since the CUFFT library has been utilized, this portion of computation is assumed to be already optimal in performance. The new GPU kernel, which performs the element-by-element multiplication of the pertinent complex matrices, has a memory-only performance of 80.57 GB/s. With a CGMA ratio of 1.5 for both SP and DP computations, the kernel is clearly memory-bound having a theoretical peak performance of 38.25 GFLOP/s in SP and 19.12 GFLOP/s in DP computations, as computed from (6.10). In the new GPU kernel, one thread block computes one sample of the response of the system. The observed performance is 26.50 GFLOP/s and 15.75 GFLOP/s in SP and DP, respectively, which is approximately 70% and 82% of the theoretically expected performance for SP and DP. Such behavior is expected because the custom GPU kernel is memory-bound and shows a memory throughput of approximately 79% of the theoretically available memory-bandwidth.

6.4.4 Example 4: Time Domain Analysis of a Nonlinear System

The fourth example considers a single degree-of-freedom dynamical system as shown in Figure 6.14. The system is assumed to have a nonlinear damping term and is subjected to a random load given by

$$f(t) = A \cdot [sin(t) + \epsilon_t] \tag{6.16}$$

where A is the amplitude of the load and ϵ_t is a unit intensity Gaussian white noise random process. The equation of motion of the dynamical system is given by

$$m\ddot{x} + c\dot{x} + \text{sgn}(\dot{x})|\dot{x}|^{0.5} + kx = f(t) \tag{6.17}$$

where $m = 2$ kg, $c = 1.6$ Ns/m, and $k = 62.84$ N/m represent the mass, the damping and the stiffness of the system, respectively. A fourth-order explicit Rünge-Kutta method has been employed to numerically integrate the equations of motion. A 10 second time history has been computed with $N_t = 1001$ discrete points for a varying number of samples, N_s, of the input.

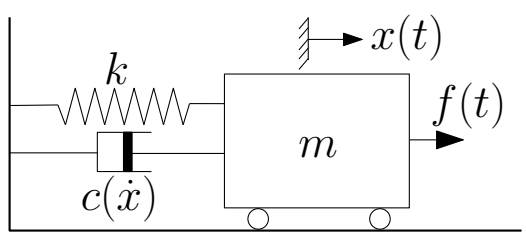


Figure 6.14: Single degree-of-freedom System

A Type I parallelization strategy and a Type 2 implementation approach has been employed. In both the GPU and the CPU implementations, each thread computes one sample of the response of the system, as shown in Figure 6.15. 'RK' denotes the Rünge-Kutta ODE solver, $f^{(i)}(t)$ is the ith sample of the random input to the system and $x^{(i)}(t)$ and $\dot{x}^{(i)}(t)$ are the corresponding displacement and velocity time-histories. T_i represents the ith parallel thread, *i.e.*, thread with index i. A pthread implementation has been utilized to distribute the work among parallel CPU threads. Table 6.5 shows the gains in computational efficiency obtained for this example given as a function of the number of samples computed. The gain in computational efficiency obtained

using the multi-core CPU implementation asymptotically approaches 8, which is the maximum achievable theoretical value as discussed earlier. The GPU implementation displays a gain in computational efficiency of approximately 2300 times as compared to the single-core CPU implementation and approximately 300 times as compared to the multi-core CPU implementation for SP computations for the largest considered problem size. The corresponding gains in computational efficiency for DP computations are observed to be approximately 140 times and 19 times, respectively.

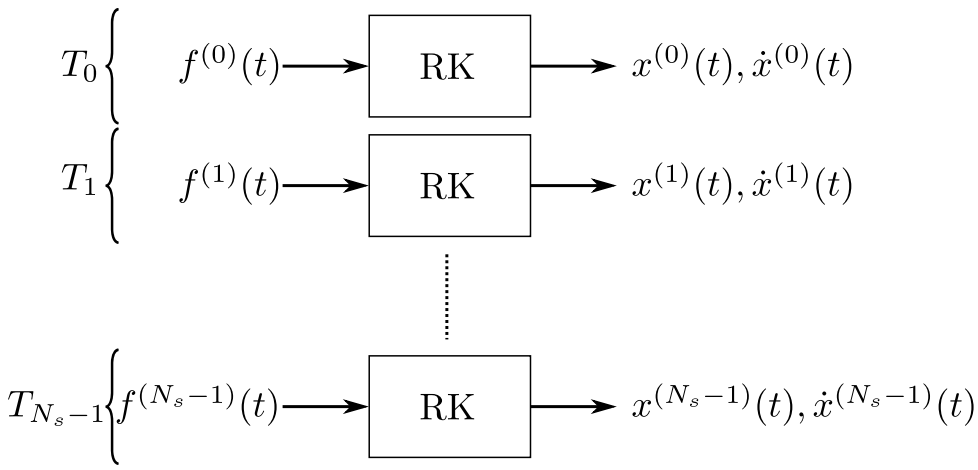


Figure 6.15: Work Division Between Threads for Example 4

Table 6.5: Optimized Gains in Computational Efficiency for Example 4

# Samples	S_{MC}	S_{GS}		S_{GM}	
		SP	DP	SP	DP
1	0.39	0.48	0.07	1.25	0.20
10	1.88	4.63	0.73	2.47	0.39
100	2.56	44.16	5.22	17.24	2.06
1000	4.39	426.93	51.80	97.27	11.94
10000	6.58	1994.32	121.78	303.09	18.83
50000	7.47	2321.44	135.79	310.72	18.57
100000	7.63	2336.58	138.08	306.37	18.37

The CGMA ratio is 49 for SP and 72 for DP. Since the number of global memory

accesses is the same for both SP and DP, the difference in the CGMA is due to the difference in the number of compute instructions. Although the same GPU kernel is utilized for both SP and DP computations, the difference in the number of compute instructions arises due to different implementations of the divide and the sine function at compiler-level. CUDA implements fast versions of floating-point division and transcendental functions (sine, cosine, *etc.*) for SP that are not available for DP thus causing the observed performance gap. Based on the CGMA ratio, it is clear that the GPU implementation is compute-bound.

The observed performance for SP and DP computations is 536.78 GFLOP/s and 51.48 GFLOP/s, respectively. These are approximately 50% of that of their respective theoretical peaks. The difference between the observed and the theoretical performance of the GPU implementation can be attributed to low processor occupancy (25% for SP and 18.8% for DP computations) and high instruction overhead. The processor occupancy is low due to high register pressure which, again, is due to a fairly large number of instructions in the GPU kernel. The size of the GPU kernel is large because each thread is used to compute one sample of the dynamic response of the system; therefore, each thread must implement the four-stage explicit Rünge-Kutta method to solve the dynamical system. All the loops have been unrolled inside the kernel in order to improve computational efficiency which also increases the size of the kernel.

In order to ensure optimal use of the GPU, a non-conventional data-structure has been utilized for the storage of the samples of the dynamic response of the system in order to guarantee that the accesses to the global memory are coalesced. An intuitive data-structure to store the response samples of the system, which has been utilized for the single- and multi-core CPU implementations is shown in Figure 6.16. The subscripts denote the time-step and the superscripts denote the sample number of the response, *i.e.*, $x_i^{(j)} = x^{(j)}(t_i)$ is the jth sample of the displacement at time t_i of the system. Adjacent cells represent contiguous memory locations.

It can be observed that each thread accesses two sets of contiguous memory locations in order to store the displacement and the velocity of the system, which are separated by N_t memory locations. As the time-marching progresses, these two memory locations shift to the right by a stride of unity. Threads with consecutive indices access memory locations that are separated by a distance $2N_t$, which is the number of memory

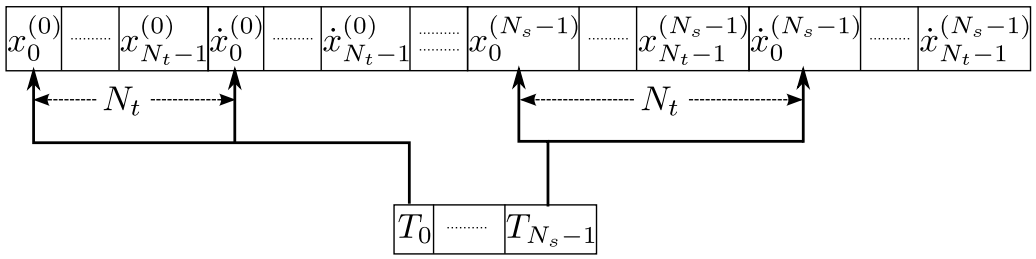


Figure 6.16: Intuitive Data-Structure for Example 4

locations required to store one sample of the entire dynamic response of the system. Thus, adjacent threads don't access adjacent memory locations. This memory access pattern doesn't affect the performance of the CPU implementations; however, in GPU implementations, this pattern is referred to as non-coalesced memory access and can cause considerable degradation in the performance of the GPU kernel.

If a similar data-structure is utilized for the GPU implementation, its memory-only performance is only 4.68 GB/s, which is considerably less than the theoretically available peak memory throughput of 102 GB/s. Consequently, the observed gains in computational efficiency are considerably reduced, as shown in Table 6.6. It can be observed that the efficiency of SP computations is diminished by an order of magnitude. If the kernel was memory-bound with a memory throughput of 4.68 GB/s, the performance of SP and DP computations would be 28.66 GFLOP/s and 42.12 GFLOP/s, respectively, as compared to 536.78 GFLOP/s and 51.48 GFLOP/s observed for the optimized kernel. This explains why the performance of DP computations is not as affected as of SP computations. Moreover, it can be observed that a non-coalesced memory access pattern can render an otherwise compute-bound kernel, memory-bound and drastically degrades its performance.

In order to ensure coalesced memory access and improve the memory throughput, a non-conventional data-structure has been utilized for the GPU implementation, as shown in Figure 6.17. Each thread accesses two contiguous memory locations to store its sample of the displacement and the velocity of the system for current time; successive threads utilize successive memory locations to store their respective samples. As the time-marching progresses, the entire access pattern is shifted towards the right by a

Table 6.6: Unoptimized Gains in Computational Efficiency for Example 4

# Samples	S_{MC}	S_{GS}		S_{GM}	
		SP	DP	SP	DP
1	0.29	0.34	0.07	1.17	0.24
10	1.82	1.26	0.71	0.69	0.39
100	2.26	9.37	5.33	4.15	2.34
1000	4.49	94.03	51.20	20.94	11.40
10000	6.54	142.60	111.60	21.80	17.06
50000	7.07	145.74	120.69	20.61	17.07
100000	7.33	147.72	122.35	20.15	16.69

stride of $2N_s$. Thus, memory coalescing is guaranteed for the entire time-marching scheme. The memory-only performance of the optimized GPU implementation is 76.4 GB/s, which is considerably better compared to the earlier 4.68 GB/s.

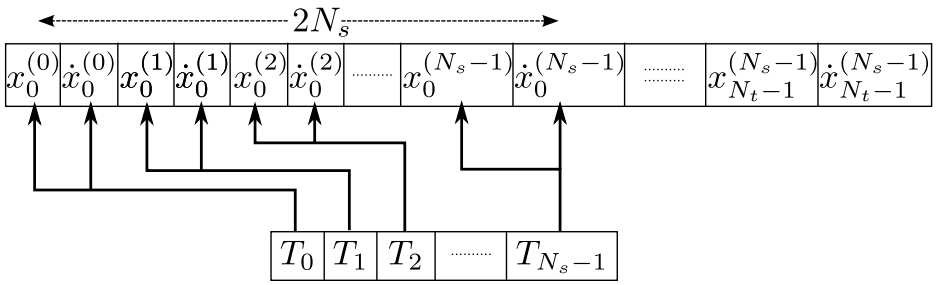


Figure 6.17: Data-Structure used for GPU implementation of Example 4

6.4.5 Example 5: Multi-dimensional Numerical Quadrature

Numerical quadrature finds many applications in uncertainty quantification procedures, especially when aleatoric uncertainties are involved. The most common application is the numerical approximation of the expectation operator. The expected value of a function, $g(\mathbf{q})$, of N random variables, $\mathbf{q} = \{q_1, \ldots, q_N\}^T$ having a joint probability

density function (PDF), $f_{\mathbf{Q}}(\mathbf{q})$, can be computed as

$$\mathbb{E}[g(\mathbf{q})] = \underbrace{\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty}}_{N} g(q_1, \ldots, q_N) f_{\mathbf{Q}}(q_1, \ldots, q_N) \mathrm{d}\mathbf{q}. \tag{6.18}$$

Although analytical integration can be performed for certain types of PDFs, *e.g.* the Gaussian PDF, numerical quadrature is unavoidable in most scenarios. In such cases, the multi-dimensional integration in (6.18) can be computed approximately using a quadrature rule as

$$\mathbb{E}[g(\mathbf{q})] \approx \underbrace{\sum_{i_1=1}^{N_a} \cdots \sum_{i_N=1}^{N_a}}_{N} w_{i_1,\ldots,i_N} g(q_{i_1}, \ldots, q_{i_N}) f_{\mathbf{Q}}(q_{i_1}, \ldots, q_{i_N}) \tag{6.19}$$

where N_a is the number of abscissas used by the quadrature rule and $w_{i_1,\ldots,i_N}$ are the associated weights. When the quadrature rule in (6.19) is constructed as a tensor-product of a one-dimensional quadrature rule, the abscissas and the weights in (6.19) can be computed using appropriate combinations of the one-dimensional abscissas and weights. Consequently, the computational complexity of the multi-dimensional numerical quadrature in (6.19) increases exponentially with the number of dimensions; in particular, the number of quadrature points in the N-dimensional space is N^{N_a}. Thus, a multi-dimensional numerical quadrature can be computationally very demanding.

In this example, Gauss-Hermite quadrature has been utilized to compute the expected value of a function of N independent, standard normal random variables. The function used in this example is given by

$$g(\mathbf{q}) = \mathbf{q}^{\mathsf{T}}\mathbf{q}. \tag{6.20}$$

Both the CPU and the GPU implementations first evaluate the integrand at the quadrature points in parallel and then perform a parallel sum-reduction to compute the approximate expected value. Four Gauss points ($N_a = 4$) in each dimension were utilized for the quadrature and a Type 2 implementation approach has been employed. OpenMP [225] has been utilized for the multi-core CPU implementation while a new CUDA kernel has been implemented.

Table 6.7 shows the gains in computational efficiency obtained for this example. For the largest considered problem size, the GPU implementation shows gains in

computational efficiency of about 50 times and 30 times for SP and DP computations respectively, as compared to the single-core CPU implementation. Compared to the multi-core CPU implementation, the corresponding gains in computational efficiency are about 10 and 5 times respectively for SP and DP computations. While the gains in computational efficiency shown by the GPU implementation are not as great as those in previous examples, it should be noted that for smaller dimensions, the multi-core CPU implementation is computationally much inferior to the single-core CPU implementation while the GPU implementation offers reasonable efficiency gains.

Table 6.7: Gains in Computational Efficiency for Example 5

# Dimensions	S_{MC}	S_{GS}		S_{GM}	
		SP	DP	SP	DP
6	0.01	1.17	1.34	403.89	149.89
7	0.02	3.42	4.13	368.69	259.41
8	0.04	15.01	8.60	345.59	204.16
9	0.16	33.76	25.07	116.11	153.53
10	1.71	39.31	37.02	19.43	21.70
11	2.74	49.44	34.00	19.23	12.41
12	4.46	56.12	27.31	14.71	6.13
13	4.75	56.59	28.90	11.76	6.08
14	5.92	54.48	27.98	10.68	4.72
15	6.16	54.43	28.11	9.60	4.56

The GPU implementation consists of two kernels. The first kernel evaluates the integrand at the quadrature points and performs a parallel sum-reduction on the data to compute the expected value. However, the parallel reduction can only be performed within a thread-block due to the fact that threads from different thread-blocks cannot communicate with each other. Therefore, a second GPU kernel is required, which performs the parallel sum-reduction on the data recursively until the data can fit within one thread-block and the approximate expected value can be computed.

The first kernel consumes about 99% of the total computational time and thus governs the performance of the GPU implementation. It achieves a memory throughput of only 0.83 GB/s due to conditional access of the global memory by the threads. The CGMA ratio changes with the number of dimensions in the first kernel because

as the number of dimensions increase, the number of arithmetic operations increase considerably while the number of global memory accesses increase only slightly.

Each thread is utilized to compute the value of the integrand at one mesh-point. Since a tensor-product multi-dimensional quadrature is being performed, the abscissas and the weights at each mesh-point are computed as the combination of appropriate abscissas and weights of the corresponding one-dimensional quadrature rule. In order to avoid conditional statements based on the thread-index to determine which abscissas and weights are to be utilized for each function evaluation, the thread-indices are converted into length-N, base-N_a bit-strings, using the first few required ASCII characters. These bit-values can be directly cast into integers that index the abscissas and the weights of the one-dimensional quadrature rule, required to compute the abscissa and the weight at a particular mesh-point of the N-dimensional space. The function-values thus computed, are stored in a shared-memory array within each thread-block, which is utilized towards the end of the first kernel to perform a preliminary parallel sum-reduction.

This idea is illustrated in Figure 6.18. A mesh is shown with $N = 3$ and $N_a = 2$. The tensor-product abscissas and the corresponding weights are shown near each mesh-point. Also shown, is the index of the thread, which is utilized to evaluate the function at that particular mesh-point. The mesh-points are assigned to the threads in a way that a length-3, base-2 bit-string of the thread index can be utilized to determine the combination of the one-dimensional quadrature rule abscissas and weights, required for the tensor-product quadrature rule. It can be seen that these bit-strings, $0_{10} = (000)_2$, $1_{10} = (001)_2$, $2_{10} = (010)_2$, *etc.*, represent the combination of the abscissas and the weights exactly, and conditional statements can be avoided.

The CGMA ratio of the GPU implementation increases with the number of dimensions, 20 for 6 dimensions to 47 for 15 dimensions, rendering the implementation compute-bound for larger dimensions. The observed computational performance is 50.61 GFLOP/s for SP and 24.26 GFLOP/s for DP computations. There are two primary sources of the sub-optimal performance of the GPU implementation: (i) the presence of a large number of thread synchronization points within the thread-blocks and (ii) the presence of conditional statements causing divergent branches of threads. Both the synchronization points and the conditional statements are necessitated by the parallel

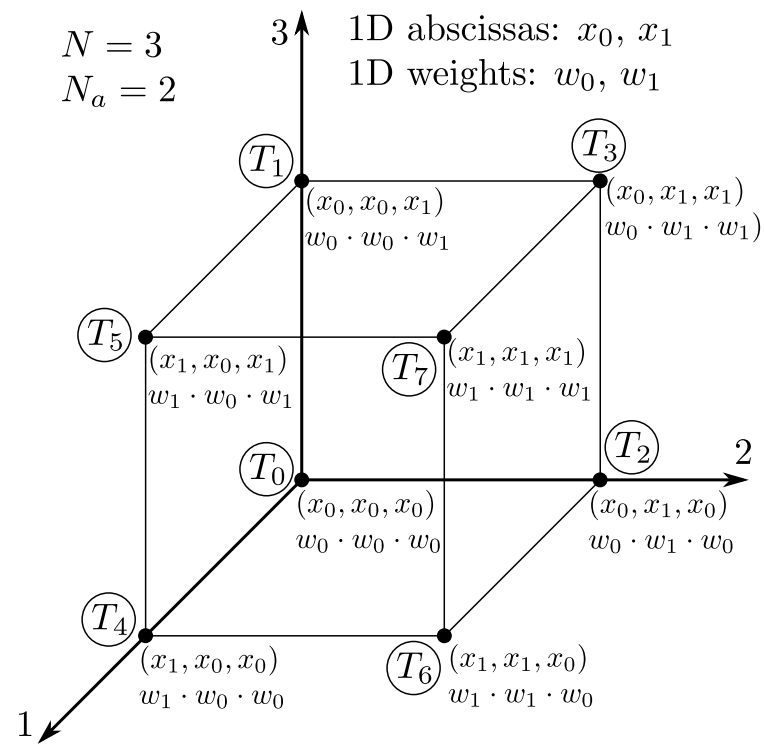


Figure 6.18: Mesh Decomposition of Parallel Threads for Example 5

sum-reduction.

6.4.6 Example 6: 3D Building Model with Tuned Mass Damper

A 612 DOF finite element model of a steel building subjected to wind excitation is considered with a tuned mass damper (TMD) on the roof as shown in Figure 5.5. Without the TMD, the structure has a 1.42 Hz fundamental natural frequency and 3% modal damping; the TMD is tuned to 1.38 Hz which gives the lowest peaks in the transfer function from wind excitation to roof displacement; the resulting combined system has a fundamental frequency of 1.30 Hz. The building is subjected to wind excitation in one direction (oriented along the TMD motion in Figure 5.5), modeled as a narrow-band filtered Gaussian white noise process centered near the fundamental mode of the structure without the TMD and shaped vertically as a power law function of height.

An NVIE-based method, as outlined by Algorithm 6 in Section 5.2, has been utilized to compute the response of the system. The computational implementation of Algorithm 6 utilizes both CPU and GPU. The computation of the forced and the impulse response of the nominal system is performed on CPU while the solution of the non-standard NVIEs is performed on GPU. Table 6.8 shows the gains in computational efficiency obtained when using a GPU accelerated implementation as compared to when using only a CPU implementation of the proposed time domain method. It can be observed that an additional gain in computational efficiency of one order of magnitude can be obtained when using a hybrid implementation that utilizes both CPU and GPU.

Table 6.8: Computational Times and Computational Efficiency for Example 6

N_s	Brute-force method computational time (s)	Proposed method (CPU implementation only)		Proposed method (with GPU acceleration)	
		Time (s)	Gain in efficiency	Time (s)	Gain in efficiency
1	15.94	32.62	0.49	31.45	0.51
10	185.50	48.50	3.82	36.89	5.03
100	2194.85	163.40	13.43	47.59	46.12
1000	22023.43	1186.52	18.56	72.20	305.05

It should be noted that the gains in computational efficiency attained by the CPU implementation of the proposed method presented in this chapter differ from those presented in the previous chapter. This is because the computational times given in Table 6.8 have been computed using a C++ implementation of the proposed method while in Chapter 5, a Matlab implementation was utilized. The computational performance of the C++ implementation of the fourth order Rünge-Kutta method, written by the author, is superior when compared to Matlab's `ode45` function. This routine is used to compute both the brute-force timings and the nominal system response used in the proposed method and is primarily responsible for the different efficiency gains observed herein.

6.5 Conclusions

This chapter first presented GPU implementations of five techniques frequently utilized in the uncertainty analysis of dynamical and mechanical systems and provided significant evidence of the potential of the use of GPUs in this area. The examples presented utilized three kinds of parallelization strategies discussed earlier in the chapter. The first four examples presented implementations of static and dynamic reanalysis techniques which can be easily adapted to other uncertainty quantification procedures, such as sensitivity analysis, model validation and calibration and design exploration and optimization. The fifth example presented the implementation of a multi-dimensional numerical quadrature scheme which is utilized in many uncertainty analysis procedures, such as sampling-based integration strategies, deterministic integration approach, sparse grid cubature, and stochastic Galerkin method. The final example considered a hybrid implementation of the efficient time domain method developed in Chapter 5. This example demonstrated that an additional gain in computational efficiency of one order of magnitude can be obtained via a combined use of the mathematical techniques developed earlier and the parallelizations strategies developed in this chapter.

The examples presented in this chapter utilized three approaches to implement a given uncertainty quantification procedure on the GPU: (i) use of pre-compiled, optimized libraries, (ii) mixed use of pre-compiled libraries with newly developed CUDA kernels, and (iii) use of completely new CUDA kernels. While using pre-compiled libraries may be the most straightforward approach to exploit the enormous parallelism offered by GPUs, it is also the least flexible among the three approaches. A major limitation of using these optimized libraries is the necessity to adapt the application to the format required by the algorithms available in the library as most of these libraries are available as pre-compiled binary files rather than actual CUDA kernels. If the implementation sought is a part of a larger application, the inability to embed the relevant computational implementation within the CUDA kernel of the larger application may hamper the overall computational performance drastically. The other two implementation approaches may require more work but yield better computational performance as was demonstrated by the examples.

The GPU implementations presented in this chapter are expected to serve as

initial bases for further developments in the use of GPUs in the field of uncertainty quantification. The specific discussions regarding the parallelization strategies and the implementation details of CUDA kernels are expected to (i) aid the understanding of the performance constraints on the relevant GPU implementations and (ii) provide some guidance regarding the computational and the data structures to be utilized in novel applications of GPU computing in the problems related to uncertainty quantification.

Chapter 7

Conclusions and Future Work

Several computationally efficient methods to perform the uncertainty quantification of large dynamical systems with local features were developed. Substantial gains in computational efficiency were observed for moderately sized computational models which are expected to be more pronounced for larger models. In addition, the usability of graphics processing units (GPUs) in the area of uncertainty quantification was explored. It was found that GPUs may provide superior parallel computing alternatives to traditional CPU-based parallel computing at least in the context of a desktop machine.

Chapter 3 presented an efficient computational methodology for the modal analysis of linear systems with local stiffness uncertainties. The proposed method assumed that the sub-spectrum of interest of a nominal system has already been computed. The relevant sub-spectrum of the related systems was computed by projecting the associated eigenvalue problems in a reduced dimensional linear generalized coordinate space. The basis of this reduced dimensional space was constructed by augmenting the sub-spectrum of the nominal system with additional enrichment vectors that utilized the knowledge about the pattern of the modifications of the related systems. Theoretical error bounds on the eigenvalues and the eigenvectors of the related systems computed using the proposed method were also discussed along with the theoretical computational costs associated with the proposed method. Gains in computational efficiency of $\mathcal{O}\left(10^2\right)$ were observed for a 612 degree-of-freedom model and that of $\mathcal{O}\left(10^3\right)$ were observed for a $17,687$ degree-of-freedom model. Future investigations will be aimed at addressing the damped eigenvalue problem and thus, include damping uncertainties in the modal

analysis as well.

Chapter 4 presented an efficient computational methodology for the estimation of the transfer function matrix of linear, time-invariant systems with local linear damping and stiffness uncertainties. The proposed method utilized the frequency response of an underlying nominal system to compute the response of a family of related systems. The principal component of the computational efficiency of the proposed method was the reduction of the size of matrices being inverted. Some characteristics of the proposed method that make it a versatile procedure that can be utilized in many application scenarios include:

- ability to accommodate non-symmetric system matrices
- ability to accommodate non-symmetric perturbations in matrices
- ability to accommodate non-classical damping
- ability to directly compute the output of only selected degrees-of-freedom
- absence of any approximations in the analytical development

The proposed method can also be utilized to compute the power spectral density of the response of a system subjected to a stationary random input process. Gains in computational efficiency of $\mathcal{O}\left(10^3\right)$ were observed for a $17,687$ degree-of-freedom computational model. Future investigations will be aimed at addressing systems subjected to non-stationary random inputs and the efficient computation of the evolutionary power spectral density of the system's response.

Chapter 5 presented a method for the rapid uncertainty quantification of the time domain response of dynamical systems with local nonlinearities and/or uncertainties in the stiffness and/or damping properties of the system. The proposed method first computed the response of a related linear system and then rapidly computed the response of an ensemble of locally modified systems as the sum of this nominal linear response and a modification term. This modification term was determined through the solution of a nonlinear Volterra integral equation written in non-standard form. A fast solution algorithm employing Newton-Gregory quadrature and a recursive convolution methodology for the governing nonlinear Volterra integral equation was

presented. Gains in computational efficiency of $\mathcal{O}\left(10^2\right)$-$\mathcal{O}\left(10^3\right)$ were observed for a $17,687$ degree-of-freedom computational model. Future efforts will be aimed at investigating other numerical methods to discretize the associated nonlinear Volterra integral equations.

Future efforts will also involve the application of the efficient computational methods developed in Chapters 3-5 to inverse problems. Moreover, these methods will be utilized to expedite the process of system identification and model validation and calibration procedures for large computational models,

Chapter 6 presented GPU implementations of five techniques frequently utilized in the uncertainty analysis of dynamical and mechanical systems and provided significant evidence of the potential of the use of GPUs in this area. The first four examples presented implementations of static and dynamic reanalysis techniques which can be easily adapted to other uncertainty quantification procedures, such as sensitivity analysis, model validation and calibration and design exploration and optimization. The fifth example presented the implementation of a multi-dimensional numerical quadrature scheme which is utilized in many uncertainty analysis procedures, such as sampling-based integration strategies, deterministic integration approach, sparse grid cubature, and stochastic Galerkin method. The final example considered a hybrid CPU-GPU implementation of the efficient time domain method developed in Chapter 5. This example demonstrated that an additional gain in computational efficiency of one order of magnitude can be obtained via a combined use of the mathematical techniques developed earlier and the parallelizations strategies developed in Chapter 6.

The GPU implementations presented in Chapter 6 showed the usefulness of GPUs in the area of uncertainty quantification and are expected to serve as initial bases for further developments in the use of GPUs in this field. The specific discussions regarding the parallelization strategies and the implementation details of CUDA kernels are expected to (i) aid the understanding of the performance constraints on the relevant GPU implementations and (ii) provide some guidance regarding the computational and the data structures to be utilized in novel applications of GPU computing in the problems related to uncertainty quantification.

The gains in efficiency presented in Chapter 6 promise to be further increased as the NVIDIA Tesla GPU utilized for this study has been superseded by the newer Fermi

GPU from NVIDIA; the Fermi offers considerable upgrades in performance compared to the Tesla. It has a larger number of computational cores, larger global and shared memory, a greater number of registers per streaming multiprocessor, higher memory bandwidth, and greatly increased double precision performance. More importantly, the shared memory of the Fermi can also be utilized as L1 cache at the discretion of the programmer, which may help hide the memory latency to a great extent. In addition, the Fermi also has error code checking (ECC) support, which improves the performance of clusters of GPUs.

The hardware development of the GPUs and their application in general purpose computing have formed a complimentary pair, with each of them driving the development of the other. This trend can be clearly observed from the hardware evolution history of the GPUs along with the footprint of the applications that utilize the computing power of the GPUs. It has been shown in this study that GPUs offer considerable advantages in uncertainty quantification problems in computational mechanics and dynamics. While an exclusive use of the GPUs may not be optimal for very large problems, there is definitely potential for the mixed use of CPUs and GPUs in this area. This use of heterogeneous computing resources in uncertainty quantification is one thrust of future investigations in this area. Other potential future investigations include (i) combining the mathematical techniques developed in Chapter 3-Chapter 5 with heterogeneous parallel computing to further enhance their computational efficiencies and (ii) exploring the use of GPU-based parallel computing clusters to address computational models of extremely large sizes.

References

[1] G. Matheron. *Estimating and choosing: an essay on probability in practice.* Springer-Verlag, Berlin, 1989.

[2] W. Heisenberg. Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik. *Zeitschrift für Physik*, 43(3-4):172–198, 1927.

[3] S.H. Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering.* Da Capo Press, 1994.

[4] R.C. Robinson. *An introduction to dynamical systems: continuous and discrete.* Prentice Hall, 2004.

[5] N. S. Abhyankar. *Studies in nonlinear structural dynamics: chaotic behavior and poynting effect.* Georgia Institute of Technology, 1987. Directed by S.V. Hanagud (U.M. order no. 86-28), 1986.

[6] J. Awrejcewicz. *Modeling, simulation and control of nonlinear engineering dynamical Systems: state-of-the-art, perspectives and applications.* Springer, December 2008.

[7] A.H. Nayfeh and P.F. Pai. *Linear and nonlinear structural mechanics.* John Wiley and Sons, 2004.

[8] W.O. Schiehlen. *Nonlinear dynamics in engineering systems: IUTAM symposium, Stuttgart, Germany, August 21-25, 1989.* Springer-Verlag, 1990.

[9] P.E. Kloeden and E. Platen. *Numerical solution of stochastic differential equations.* Springer, 1999.

[10] P.D. Spanos. *Computational stochastic mechanics: proceedings, Third International Conference on Computational Stochastic Mechanics, Thera-Santorini, Greece, 14-17 June 1998*. Taylor & Francis, April 1999.

[11] Y-K. Lin and A. der Kiureghian. *Methods of stochastic structural dynamics*. Dipartimento di Meccanica Strutturale, Università di Pavia, 1986.

[12] G.D. Manolis and P.K. Koliopoulos. *Stochastic structural dynamics in earthquake engineering*. WIT Press, May 2001.

[13] S.T. Ariaratnam, G.I. Schuëller, and I. Elishakoff. *Stochastic structural dynamics: progress in theory and applications*. CRC Press, September 1988.

[14] H. Benaroya. Stochastic structural dynamics: a theoretical basis and a selective review of the literature. Technical Report 1, Weidlinger & Associates, NY, April 1984.

[15] Y-K. Lin and G.I. Schuëller. Stochastic structural dynamics. In *International Association of Structural Safety and Reliability*. Elsevier Applied Science, 1995.

[16] G.I. Schuëller and M. Shinozuka, editors. *Stochastic methods in structural dynamics*. Springer, November 1987.

[17] J. Li and J. Chen. *Stochastic dynamics of structures*. John Wiley and Sons, May 2009.

[18] J-Q. Sun. *Stochastic dynamics and control*. Elsevier, 2006.

[19] M. Gundlach. *Stochastic dynamics*. Springer, March 1999.

[20] Y-K. Lin and G-Q. Cai. *Probabilistic structural dynamics: advanced theory and applications*. McGraw-Hill Professional, 2004.

[21] W. Kliemann, W.F. Langford, and N.S. Namachchivaya. *Nonlinear dynamics and stochastic mechanics*. AMS Bookstore, 1996.

[22] A.I. Mees. *Nonlinear dynamics and statistics*. Springer, 2001.

[23] N. Metropolis. The beginning of the Monte Carlo method. *Los Alamos Science*, (1987 special issue dedicated to S. Ulam):125–130, 1987.

[24] G.S. Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer-Verlag, New York, 1996.

[25] J.M. Hammersley and D.C. Handscomb. *Monte Carlo methods*. Wiley, New York, 1964.

[26] J. Neyman. On the two different aspects of the representative method: The method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, 97(4):558–625, 1934.

[27] M.D. McKay, W.J. Conover, and R.J. Beckman. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[28] M. Stein. Large sample properties of simulations using Latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

[29] W-L. Loh. On Latin hypercube sampling. *The Annals of Statistics*, 24(5):2058–2080, 1996.

[30] B. Efron. The Jackknife, the Bootstrap, and other resampling plans. CBMS-NSF Regional Conference Series in Applied Mathematics 38, SIAM, Philadelphia, PA, 1982.

[31] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. SIAM, Philadelphia, Pennsylvania, 1992.

[32] H. Niederreiter, P. Hellekalek, G. Larcher, and P. Zinterhof. *Monte Carlo and quasi-Monte Carlo methods 1996*. Springer-Verlag, 1998.

[33] B.L. Fox. *Strategies for quasi-Monte Carlo*. Kluwer Academic, Norwell, Massachusetts, 1999.

[34] S. Engelund and R. Rackwitz. A benchmark study on importance sampling techniques in structural reliability. *Structural Safety*, 12(4):255–276, 1993.

[35] W.R. Gilks, S. Richardson, and D.J. Spiegelhalter, editors. *Markov chain Monte Carlo in practice*. Chapman-Hall, 1996.

[36] A. Samant and D.G. Vlachos. Overcoming stiffness in stochastic simulation stemming from partial equilibrium: A multiscale Monte Carlo algorithm. *Journal of Chemical Physics*, 123(14):144114–144114–8, October 2005.

[37] A.M. Ferrenberg and R.H. Swendsen. New Monte Carlo technique for studying phase transitions. *Physical Review Letters*, 61(23):2635–2638, December 1988.

[38] A. Chatterjee and D.G. Vlachos. Multiscale spatial Monte Carlo simulations: Multigriding, computational singular perturbation, and hierarchical stochastic closures. *Journal of Chemical Physics*, 124(6):064110–064110–16, February 2006.

[39] R.L. Iman and M.J. Shortencarier. A FORTRAN 77 program and user's guide for the generation of Latin hypercube samples for use with computer models. Technical Report NUREG/CR-3624, SAND83-2365, Sandia National Laboratories, Albuquerque, NM, 1984.

[40] J.C. Helton and Davis F.J. Sampling-based methods for uncertainty and sensitivity analysis. Technical Report SAND99-2240, Sandia National Laboratories, Albuquerque, NM, 2000.

[41] K. Esselink, L. D. J. C. Loyens, and B. Smit. Parallel Monte Carlo simulations. *Physical Review E*, 51(2):1560–1568, February 1995.

[42] J.S. Rosenthal. Parallel computing and Monte Carlo algorithms. *Far East Journal of Theoretical Statistics*, 4:207–236, 2000.

[43] N.C. Nigam. *Introduction to random vibrations*. MIT Press, Cambridge, Massachusetts, 1983.

[44] T.T. Soong and M. Grigoriu. *Random vibration of mechanical and structural systems*. Englewood Cliffs, N.J., PTR Prentice Hall, 1993.

[45] A.H. Nayfeh. *Perturbation methods*. John Wiley & Sons, New York, 1973.

[46] W.K. Liu, T. Belytschko, and A. Mani. Random field finite elements. *International Journal of Numerical Methods in Engineering*, 23:1831–1845, 1986.

[47] W.K. Liu, T. Belytschko, and A. Mani. Probabilistic finite elements for nonlinear structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 56(1):61–81, 1986.

[48] M. Kleiber and T.D. Hien. *The stochastic finite element method*. John Wiley & Sons, 1992.

[49] H-P. Chen. Nonlinear perturbation theory for structural dynamic systems. *AIAA Journal*, 43(11):2412–2421, 2005.

[50] M. Shinozuka and G. Deodatis. Response variability of stochastic finite element systems. *Journal of Engineering Mathematics*, 114(3):499–519, 1988.

[51] F. Yamazaki, M. Shinozuka, and G. Dasgupta. Neumann expansion for stochastic finite element analysis. *Journal of Engineering Mathematics*, 114(8):1335–1354, 1988.

[52] G. Deodatis. Weighted integral method. I: Stochastic stiffness matrix. *Journal of Engineering Mechanics*, 117(8):1851–1864, 1991.

[53] G. Deodatis and M. Shinozuka. Weighted integral method. II: Response variability and reliability. *Journal of Engineering Mechanics*, 117(8):1865–1877, 1991.

[54] K. Karhunen. Über lineare methoden in der wahrscheinlichkeitsrechnung. *Ann. Acad. Sci. Fennicae. Ser. A, I*, 37:3–79, 1947.

[55] M. Loève. *Probability theory*. Springer-Verlag, 4 edition, 1978.

[56] R.G. Ghanem and P.D. Spanos. *Stochastic finite elements: a spectral approach*. Springer-Verlag, New York, 1991.

[57] N. Wiener. The homogeneous chaos. *American Journal of Mathematics*, 60:897–936, 1938.

[58] D. Xiu. *Numerical methods for stochastic computations: a spectral method approach*. Princeton University Press, Princeton, 2010.

[59] O.P. Le Maître and O.M. Knio. *Spectral methods for uncertainty quantification: with applications to computational fluid dynamics.* Springer, New York, 2010.

[60] R. Li and R. Ghanem. Adaptive polynomial chaos expansions applied to statistics of extremes in nonlinear random vibration. *Probabilistic Engineering Mechanics*, 13:125–136, 1998.

[61] R. Ghanem. Hybrid stochastic finite elements and generalized Monte Carlo simulation. *ASME J. Appl. Mech.*, 65:1004–1009, 1998.

[62] D. Xiu and G. Karniadakis. Modeling uncertainty in steady state diffusion problems via generalized polynomial chaos. *Comput. Methods Appl. Mech. Engrg.*, 191:4927–4948, 2002.

[63] D. Xiu and G. Karniadakis. Modeling uncertainty in flow simulations via generalized polynomial chaos. *Journal of Computational Physics*, 187:137–167, 2003.

[64] M. Raegan, H. Najm, P. Pébay, O. Knio, and R. Ghanem. Quantifying uncertainty in chemical systems modeling. *Int. J. Chem. Kinet.*, 37:368–382, 2005.

[65] G.N. Fortuna and A. Gallo. *Model order reduction techniques with applications in electrical engineering.* Springer-Verlag, London, 1992.

[66] Z-Q. Qu. *Model order reduction techniques: with applications in finite element analysis.* Springer-Verlag, London, August 2004.

[67] A.C. Antoulas. *Approximation of large-scale dynamical systems.* SIAM, Philadelphia, 2005.

[68] W.H.A. Schilders, H.A. van der Vorst, and J. Rommes. *Model order reduction: theory, research aspects and applications.* Springer-Verlag, Berlin, 2008.

[69] E.J. Davison. A method for simplifying linear dynamic systems. *IEEE Transactions on Automatic Control*, 11(1):93–101, 1966.

[70] A. Varga. Enhanced modal approach for model reduction. *Mathematical Modelling of Systems*, 1:91–105, 1995.

[71] E.J. Grimme. *Krylov projection methods for model reduction*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[72] Z. Bai. Krylov subspace techniques for reduced-order modeling of large-scale dynamical systems. *Applied Numerical Mathematics*, 43:9–44, 2002.

[73] R.W. Freund. Model reduction methods based on Krylov subspaces. *Acta Numerica*, 1:267–319, 2003.

[74] S.B. Salimbahrami. *Structure preserving order reduction of large scale second order models*. PhD thesis, Technische Universität München, 2005.

[75] R. Eid, B. Salimbahrami, and B. Lohmann. Parametric order reduction of proportionally damped second order systems. Technical Report TRAC-1, Institute of Automatic Control, Technical University of Munich, October 2006.

[76] P. Feldmann. Model order reduction techniques for linear systems with large numbers of terminals. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 2004.

[77] P. Benner and A. Schneider. Model order and terminal reduction approaches via matrix decomposition and low rank approximation, October 2008.

[78] A. Chatterjee. An introduction to the proper orthogonal decomposition. *Current Science*, 78(7):808–817, April 2000.

[79] W. Gressick, J.T. Wen, and J. Fish. Order reduction of large-scale finite element models: A systems perspective. *Intl. J. for Multiscale Computational Engineering*, 3(3):337–362, 2005.

[80] S. Gugercin and K. Willcox. Krylov projection framework for Fourier model reduction. *Automatica*, 44(1):209–215, 2008.

[81] R. Genesio and M. Milanese. A note on the derivation and use of reduced-order model. *IEEE Transactions on Automatic Control*, 21:118–122, 1976.

[82] J. Hickin and N. Sinha. Model reduction for linear multivariable systems. *IEEE Transactions on Automatic Control*, 25(6):1121–1127, 1980.

[83] J.R. Phillips. Projection-based approaches for model reduction of weakly nonlinear, time-varying systems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(2):171–187, 2003.

[84] M.J. Rewieński. *A trajectory piecewise-linear approach to model order reduction of nonlinear dynamical systems.* PhD thesis, Massachusetts Institute of Technology, 2003.

[85] J. Chen, S-M. Kang, J. Zou, C. Liu, and J.E. Schutt-Aine. Reduced-order modeling of weakly nonlinear MEMS devices with Taylor-series expansion and Arnoldi approach. *Journal of Microelectromechanical Systems*, 13(3):441–451, 2004.

[86] D.J. Lucia, P.S. Beran, and W.A. Silva. Reduced-order modeling: new approaches for computational physics. *Progress in Aerospace Sciences*, 40(1-2):51–117, February 2004.

[87] L. Feng. Review of model order reduction methods for numerical simulation of nonlinear circuits. *Applied Mathematics and Computation*, 167(1):576–591, August 2005.

[88] J. S. Arora. Survey of structural reanalysis techniques. *J. of the Structural Division (ASCE)*, 102(4):783–802, 1976.

[89] A. A. M. Kasim. Static reanalysis: a review. *J. of the Structural Divisiion (ASCE)*, 113(6):1029–1045, 1987.

[90] J.A. Brandon. *Strategies for structural dynamic modification.* Research Studies Press Ltd., Wiley, 1990.

[91] S. H. Chen and X. W. Yang. Extended Kirsch combined method for eigenvalue analysis. *AIAA J.*, 38(5):927–930, 2000.

[92] S. H. Chen, X. W. Yang, and H. D. Lian. Comparison of several eigenvalue reanalysis methods for modified structures. *Structural Optimization*, 20(4):253–259, 2000.

[93] S. H. Chen. *Matrix perturbation theory in structural dynamics.* Intl. Academic Publisher, Beijing, 1993.

[94] W. Bickford. An improved computational technique for perturbations of the generalized symmetrical linear algebraic eigenvalue problem. *International Journal for Numerical Methods in Engineering*, 24(3):529–541, 1987.

[95] S. Chen, D. Song, and A. Ma. Eigensolution reanalysis of modified structures using perturbations and Rayleigh quotients. *Communications in Numerical Methods in Engineering*, 10(2):111–119, 1994.

[96] B. Cochelin, N. Damil, and M. Potierferry. Asymptotic numerical-methods and Pade approximants for nonlinear elastic structures. *International Journal for Numerical Methods in Engineering*, 37(7):1187–1213, 1994.

[97] J. Hurtado. Reanalysis of linear and nonlinear structures using iterated shanks transformation. *Computer Methods in Applied Mechanics and Engineering*, 19:4215–4229, 2002.

[98] Hua-Peng Chen. Efficient methods for determining modal parameters of dynamic structures with large modifications. *Journal of Sound and Vibration*, 298(1-2):462–470, November 2006.

[99] U. Kirsch. Approximate vibration reanalysis of structures. *AIAA J.*, 41:504–511, 2003.

[100] U. Kirsch and M. Bogomolni. Procedures for approximate eigenproblem reanalysis of structures. *International Journal for Numerical Methods in Engineering*, 60(12):1969–1986, 2004.

[101] U. Kirsch and M. Bologami. Error evaluation in approximate reanalysis of structures. *Structural and Multidisciplinary Optimization*, 28(2–3):10, 2004.

[102] U. Kirsch, M. Bogomolni, and I. Sheinman. Efficient dynamic reanalysis of structures. *Journal of Structural Engineering, ASCE*, 3:440–448, 2007.

[103] Y. Yasui. Direct coupled load verification of modified structural component. *AIAA J.*, 36(1):94–101, 1998.

[104] J.K. Liu and H.C. Chan. A universal matrix perturbation technique for structural dynamic modification using singular value decomposition. *Journal of Sound and Vibration*, 228(2):265–274, 1999.

[105] E. Balmès. Optimal Ritz vectors for component mode synthesis using the singular value decomposition. *AIAA J.*, 34(6):1256–1260, 1996.

[106] G. Zhang. *Component-based and parametric reduced-order modeling methods for vibration analysis of complex structures*. PhD thesis, University of Michigan, Ann Arbor, MI, 2005.

[107] S.-K. Hong, B.I. Epureanu, M.P. Castanier, and D.J. Gorsich. Parametric reduced-order models for predicting the vibration response of complex structures with component damage and uncertainties. *Journal of Sound and Vibration*, 330(6):1091–1110, 2011.

[108] G. Zhang, E. Nikolaidis, and Z.P. Mourelatos. An efficient re-analysis methodology for probabilistic vibration of large-scale structures. *Journal of Mechanical Design*, 131(5):051007, 2009.

[109] S.F. Wojtkiewicz and Gaurav. Efficient modal analysis of structures with local stiffness uncertainties. *International Journal for Numerical Methods in Engineering*, 80:1007–1024, 2009.

[110] E. Wilson, M. Yuan, and J. Dickens. Dynamic analysis by direct superposition of Ritz vectors. *Earthquake Engineering and Structural Dynamics*, 10:813–821, 1982.

[111] H. Nasr. An efficient frequency response computation using Arnoldi's algorithm. *International Journal of Computer Mathematics*, 46(3):183–194, 1992.

[112] W.E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9:17–29, 1951.

[113] C.-W. Kim and J.K. Bennighof. Fast frequency response analysis of partially damped structures with non-proportional viscous damping. *Journal of Sound and Vibration*, 297(3-5):1075–1081, 2006.

[114] C.-W. Kim and J. K. Bennighof. Fast frequency response analysis of large-scale structures with non-proportional damping. *International Journal for Numerical Methods in Engineering*, 69(5):978–992, 2007.

[115] C.-W. Kim. Damped dynamic response determination in the frequency domain for partially damped large scale structures. *Journal of Sound and Vibration*, 326(3-5):703–708, 2009.

[116] J.K. Bennighof and M.F. Kaplan. Frequency sweep analysis using multi-level substructuring, global modes and iteration. In *Proceedings of 39th AIAA/ASME/ASCE/- AHS Structures, Structural Dynamics and Materials Conference*, pages 98–2015, 1998.

[117] J.H. Ko and Z. Bai. High-frequency response analysis via algebraic substructuring. *International Journal for Numerical Methods in Engineering*, 76(3):295–313, 2008.

[118] J. Ko, D. Byun, and J. Han. An efficient numerical solution for frequency response function of micromechanical resonator arrays. *Journal of Mechanical Science and Technology*, 23(10):2694–2702, 2009.

[119] S.F. Wojtkiewicz, Gaurav, and Q.I. Odes. Efficient frequency response of locally uncertain linear structural systems. *Journal of Engineering Mechanics, ASCE*, 137(2):147–150, 2011.

[120] Gaurav and S.F. Wojtkiewicz. Efficient spectral response of locally uncertain linear systems. *Probabilistic Engineering Mechanics*, 25(4):419–424, 2010.

[121] J. Sherman and W.J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[122] M.A. Woodbury. Inverting modified matrices. *Memorandum report*, 42:106, 1950.

[123] K.E. Rouch and J.S. Kao. Dynamic reduction in rotor dynamics by the finite element method. *Journal of Mechanical Design*, 102(2):360–368, 1980.

[124] L.J. McLean and E.J. Hahn. Unbalanced behavior of squeeze film damped multi-mass flexible rotor bearing systems. *Journal of Lubrication Technology*, 105(1):22–28, 1983.

[125] J.L. Huang and T.N. Shiau. An application of the generalized polynomial expansion in method to nonlinear rotor bearing systems. *ASME J. Vibration & Acoustics*, 113:299–308, 1994.

[126] R.H.B. Fey, D.H. van Campen, and A. de Kraker. Long term structural dynamics of mechanical systems with local nonlinearities. *ASME J. Vibration & Acoustics*, 118:147–153, 1996.

[127] T. Zheng and N. Hasebe. An efficient analysis of high-order dynamical system with local nonlinearity. *ASME J. Vibration & Acoustics*, 121:408–416, 1999.

[128] Z.-Q. Qu. Model reduction for dynamical systems with local nonlinearities. *AIAA Journal*, 40(2):327–333, 2002.

[129] D. Segalman. Model reduction of systems with localized nonlinearities. Technical Report SAND2006-1789, Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185-0557, March 2006.

[130] R.W. Clough and Wilson E.L. Dynamic analysis of large structural systems with local nonlinearities. *Computer Methods in Applied Mechanics and Engineering*, 17/18:107–129, 1978.

[131] P. Hagedorn and W. Schramm. On the dynamics of large systems with localized nonlinearities. *ASME Journal of Applied Mechanics*, 55(4):946–951, 1988.

[132] I.F. Chiang and S.T. Noah. Convolution approach for the transient analysis of locally nonlinear rotor systems. *ASME Journal of Applied Mechanics*, 57(3):731–737, 1990.

[133] Y. Iwata, H. Sato, and T. Komatsuzaki. Analytical method for steady state vibration of system with localized non-linearities using convolution integral and Galerkin method. *Journal of Sound and Vibration*, 262:11–23, 2003.

[134] J.H. Gordis and J. Radwick. Efficient transient analysis for large locally nonlinear structures. *Shock and Vibration*, 6(1):1–9, 1999.

[135] A.V. Jarque. Recursive block-by-block integral equation solution for transient dynamic analysis with memory-type elements. Master's thesis, Naval Postgraduate School, Monterey, CA, 2001.

[136] J.H. Gordis and B. Neta. Fast transient analysis for locally nonlinear structures by recursive block convolution. *Journal of Vibration and Acoustics*, 123:545–547, 2001.

[137] Q.I. Odes. Fast computational method for low-rank stiffness and damping modification in structural systems. Master's thesis, University of Minnesota, 2007.

[138] S.F. Wojtkiewicz. Ensemble uncertainty quantification. Abstracts from the 9th USNCCM, San Francisco, July 2007.

[139] P. Linz. *Analytical and numerical methods for Volterra equations*. SIAM Studies in Applied Mathematics, 1987.

[140] E. Hairer, C. Lubich, and M. Schlichte. Fast numerical solution of nonlinear Volterra convolution equations. *SIAM J. Sci. Stat. Comput.*, 6(3):532–541, 1985.

[141] H. Brunner. *Collocation methods for Volterra integral and related functional equations*. Cambridge University Press, NY, 2004.

[142] S.A. Isaacson and R.M Kirby. Numerical solution of linear Volterra integral equations of the second kind with sharp gradients. *Journal of Computational and Applied Mathematics*, 235(14):4283–4301, 2011.

[143] G. Capobianco, D. Conte, I. del Prete, and E. Russo. Fast Rünge-Kutta methods for nonlinear convolution systems of Volterra integral equations. *BIT Numerical Mathematics*, 47:259–275, 2007.

[144] A. Talbot. The accurate numerical inversion of Laplace transforms. *IMA Journal of Applied Mechanics*, 23(1):97–120, 1979.

[145] Gaurav, S.F. Wojtkiewicz, and E.A. Johnson. Efficient uncertainty quantification of dynamical systems with local nonlinearities and uncertainties. *Probabilistic Engineering Mechanics*, tentatively accepted, under revision:August, 2011.

[146] G.M. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. *AFIPS Conference Proceedings*, 30:483–485, 1967.

[147] J.L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[148] M.J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, December 1966.

[149] P. Colella. Defining software requirements for scientific computing. DARPA HPCS Presentation, 2004.

[150] K. Asanovic, R. Bodik, B.C. Catanzaro, J.J. Gebis, P. Husbands, K. Keutzer, D.A. Patterson, W.L. Plishker, J. Shalf, S.W. Williams, and K.A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report USB/EECS-2006-183, University of California at Berkeley, December 2006.

[151] S.F. Wojtkiewicz, M.S. Eldred, and R.V. Field. Uncertainty quantification in large computational engineering models. In *In Proceedings of the 42rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, number AIAA-2001-1455. Citeseer, 2001.

[152] DAKOTA. DAKOTA. http://dakota.sandia.gov/software.html, 2011.

[153] NVIDIA. NVIDIA Programming Guide, v4.0, 2011.

[154] G.E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[155] Intel. Intel Xeon quad-core e5530. http://ark.intel.com/Product.aspx?id=37103, 2010.

[156] NVIDIA. NVIDIA Tesla C1060. http://www.nvidia.com/object/product_tesla_c1060_us.html, 2010.

[157] NVIDIA. CUBLAS User's Manual, v3.1, 2010.

[158] NVIDIA. CUFFT User's Manual, v3.1, 2010.

[159] S. Tomov, J. Dongarra, and M. Baboulin. Towards dense linear algebra for hybrid gpu accelerated manycore systems. Computer Science Technical Report (also LAPACK working note 210) UT-CS-08-632, University of Tennessee, October 2008.

[160] MAGMA. Matrix Algebra on GPU and Multicore Architectures. http://icl.cs.utk.edu/magma/index.html, 2010.

[161] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In *ACM SIGGRAPH 2003 Papers*, page 924. ACM, 2003.

[162] N. Bell and M. Garland. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, December 2008.

[163] F. Vázquez, E.M. Garzón, A. Martínez, and J.J. Fernández. The sparse matrix vector product on gpus. Technical Report CSTN-077, University of Almeria, June 2009.

[164] S. Tomov, M. McGuigan, R. Bennett, G. Smith, and J. Spiletic. Benchmarking and implementation of probability-based simulations on programmable graphics cards. *Computers & Graphics*, 29(1):71–80, 2005.

[165] X. Tian and K. Benkrid. Mersenne twister random number generation on FPGA, CPU and GPU. pages 460 – 464, San Francisco, CA, United states, 2009.

[166] Poman P. M. So. Time-domain computational electromagnetics algorithms for GPU based computers. In *EUROCON 2007, The International Conference on "Computer as a Tool"*, pages 1–4, Warsaw, September 2007. IEEE.

[167] Y. Zhao. Lattice boltzmann based PDE solver on the GPU. *The Visual Computer*, 24(5):323–333, 2008.

[168] S.D.C. Walsh, M.O. Saar, P. Bailey, and D.J. Lilja. Accelerating geoscience and engineering system simulations on graphics hardware. *Computer & Geosciences*, 35:2353–2354, 2009.

[169] M. Januszewski and M. Kostur. Accelerating numerical solution of stochastic differential equations with CUDA. *Computer Physics Communications*, 181:183–188, 2009.

[170] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, and T.J. Purcell. A survey of general-purpose computation on graphics hardware. In *Computer Graphics Forum*, volume 26, pages 80–113. John Wiley & Sons, 2007.

[171] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[172] J. Cohen and M. Garland. Novel architectures: Solving computational problems with GPU computing. *Computing in Science and Engineering*, 11(5):58–63, 2009.

[173] J. Nickolls and W.J. Dally. The GPU computing era. *IEEE Micro*, 30(2):56–69, 2010.

[174] A.R. Brodtkorb, C. Dyken, T.R. Hagen, J.M. Hjelmervik, and O.O. Storaasli. State-of-the-art in heterogeneous computing. *Scientific Programming*, 18(1):1–33, 2010.

[175] NVIDIA. NVIDIA CUDA zone. http://www.nvidia.com/object/cuda_apps_flash_new.html, 2010.

[176] GPGPU. GPGPU. http://gpgpu.org, 2010.

[177] Gaurav and S.F. Wojtkiewicz. Use of GPU computing for uncertainty quantification in computational mechanics: A case study. *Scientific Programming*, accepted, 2011.

[178] Lord Rayleigh. *The theory of sound*. Dover Publications, 1945.

[179] T.K. Caughey and M.E.J. O'Kelly. Classical normal modes in damped linear dynamic systems. *Journal of Applied Mechanics*, pages 583–588, 1964.

[180] F.E. Udwadia and R.S. Esfandiari. Nonclassically damped dynamic systems an iterative approach. *Journal of Applied Mechanics*, 57(2):423–433, 1990.

[181] S.M. Shahruz and A.K. Packard. Approximate decoupling of weakly non-classically damped linear second-order systems under harmonic excitations. *Journal of Dynamic Systems*, 115(1):214–218, 1993.

[182] K. Ogata. *Modern control engineering*. Prentice Hall, Englewood Cliffs, N.J., 1970.

[183] G.F. Franklin, J.D. Powell, and A. Emami-Naeini. *Feedback control of dynamic systems*. Pearson, 2009.

[184] S. Gutman. Uncertain dynamical systems - a Lyapunov min-max approach. *Automatic Control, IEEE Transactions on*, 24(3):437–443, June 1979.

[185] K. Zhou and P. Khargonekar. Stability robustness bounds for linear state-space models with structured uncertainty. *Automatic Control, IEEE Transactions on*, 32(7):621–623, July 1987.

[186] H. Wu and K. Mizukami. Exponential stability of a class of nonlinear dynamical systems with uncertainties. *Systems & Control Letters*, 21(4):307–313, 1993.

[187] J.R. Fisher. *Stability Analysis and Control of Stochastic Dynamic Systems using Polynomial Chaos*. PhD thesis, Texas A&M University, 2008.

[188] C. Rünge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46:167–178, 1895.

[189] W. Kutta. Beitrag zur näherungsweisen integration totaler differentialgleichungen. *Zeitschrift für Mathematik und Physik*, 46:435–453, 1901.

[190] J.C. Butcher. *Numerical methods for ordinary differential equations*. Wiley, 2008.

[191] N.M. Newmark. A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division, ASCE*, 85:67–94, 1959.

[192] K.J. Bathe and E.L. Wilson. Stability and accuracy analysis of direct integration methods. *Earthquake Engineering & Structural Dynamics*, 1:283–291, 1973.

[193] C.W. Bert. Comparative evaluation of six different numerical integration methods for non-linear dynamic systems. *Journal of Sound and Vibration*, 127(2):221–229, 1988.

[194] V. Volterra. *Theory of functionals and of integral and integro-differential equations*. Blackie & Sons Limited, London and Glasgow, 1930.

[195] E.A. Coddington and N. Levinson. *Theory of ordinary differential equations*. McGraw-Hill, 1955.

[196] G.H. Golub and C.F. Van Loan. *Matrix computations*. The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[197] E. Balmès, F. Ravary, and D. Langlais. Uncertainty propagation in modal analysis. In *Proceedings - 24th International Modal Analysis Conference (IMAC-XXIV)*, Orlando, FL, 2005.

[198] R.L. Fox and M.P. Kapoor. Rates of changes of eigenvalues and eigenvectors. *AIAA J.*, 6:2426–2429, 1968.

[199] E. Balmès. Parametric families of reduced finite element models. theory and applications. *Mechanical Systems and Signal Processing*, 10(4):381–394, 1996.

[200] K.A. Kline. Dynamic analysis using a reduced basis of exact modes and Ritz vectors. *AIAA J.*, 24(12):2022–2029, 1986.

[201] C.C. Chu and M.H. Milman. Eigenvalue error analysis of viscously damped structures using a Ritz reduction method. *AIAA J.*, 30(12):2935–2944, 1992.

[202] Y. Saad. *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992.

[203] Yitshak M. Ram, Joab J. Blech, and Simon G. Braun. Eigenproblem error bounds with application to symmetric dynamic system modification. *SIAM Journal on Matrix Analysis and Applications*, 11(4):553–564, October 1990.

[204] D.J. Thompson and A.E. Schultz. Development of an advanced structural monitoring system. Draft final report, University of Minnesota, September 2010.

[205] SAP2000. SAP2000. http://www.sap2000.org, 2011.

[206] A.J. Gastineau, S.F. Wojtkiewicz, and A.E. Schultz. Fatigue life extension of vulnerable steel bridges using a response modification approach. In *EURODYN2011*, 2011.

[207] J.H. Welsch. Abscissas and weights for Gregory quadrature. *Communications of the ACM*, 9(4):271, 1966.

[208] O.E. Bringham. *Fast Fourier transform and Its applications.* Prentice-Hall, Inc., NJ, 1988.

[209] S. Kapur and V. Rokhlin. High-order corrected trapezoidal quadrature rules for singular functions. *SIAM J. Numer. Anal.*, 34(4):1331–1356, 1997.

[210] AASHTO. LRFD bridge design specifications, 2009.

[211] H.M. Hilber, T.J.R. Hughes, and R.L. Taylor. Improved numerical dissipation for time integration algorithms in structural dynamics. *Earthquake Engineering & Structural Dynamics*, 5:283–292, 1977.

[212] P.-H. Lin, J. Jayraj, and P.R. Woodward. A strategy for automatically generating high performance CUDA code for a GPU accelerator from a specialized fortran code expression. saahpc.ncsa.illinois.edu/10/papers/paper_24.pdf, 2010.

[213] T.B. Jablin, P. Prabhu, J.A. Jablin, N.P. Johnson, S.R. Beard, and D.I. August. Automatic CPU-GPU communication management and optimization. In *PLDI'11*, San Jose, CA, June 2011.

[214] A. Klöckner. PyCUDA. http://documen.tician.de/pycuda, 2011.

[215] BLAS. Basic Linear Algebra Subprograms. http://www.netlib.org/blas, 2010.

[216] NVIDIA. CUSPARSE User's Manual, 2011.

[217] FFTW. Fastest Fourier Transform in the West. http://www.fftw.org, 2010.

[218] NVIDIA. CURAND User's Guide, 2011.

[219] NVIDIA. CURAND User's Guide, v4.0, 2011.

[220] NVIDIA. Thrust quick start guide, v4.0, 2011.

[221] LAPACK. Linear Algebra PACKage. http://www.netlib.org/lapack, 2010.

[222] EM Photonics. CULA tools. http://www.culatools.com, 2011.

[223] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to parallel computing*. Pearson Education Limited, 2003.

[224] Intel. Math Kernel Library. http://software.intel.com/en-us/intel-mkl, 2010.

[225] OpenMP. Openmp. http://www.openmp.org, 2010.

Appendix A

Acronyms and List of Symbols

Care has been taken in this dissertation to minimize the use of acronyms, but this cannot always be achieved. This appendix contains a table of acronyms and the mathematical symbols used in the text.

A.1 Acronyms

Table A.1: Acronyms

Acronym	Meaning
AASHTO	American Association of State Highway and Transportation Officials
AE	Algebraic equation
ALU	Arithmetic Logic Unit
AR	Auto Regressive
ASCII	American Standard Code for Information Interchange
CGMA	Compute to Global Memory Access
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DOF	Degree-Of-Freedom
DP	Double Precision
DRAM	Dynamic Random Access Memory
ECC	Error Code Checking
FFT	Fast Fourier Transform
FLOP	Floating Point Operation

Continued on next page

Table A.1 – continued from previous page

Acronym	Meaning
GPU	Graphics Processing Unit
HHT	Hilber-Hughes-Taylor
IE	Integral Equation
MIMD	Multiple-Instruction Multiple-Data
MISD	Multiple-Instruction Single-Data
NVIE	Nonlinear Volterra Integral Equation
LDRV	Load-Dependent Ritz Vector
LTI	Linear Time-Invariant
ODE	Ordinary Differential Equation
PCI	Peripheral Component Interconnect
PDE	Partial Differential Equation
PDF	Probability Density Function
PSD	Power Spectral Density
RAM	Random Access Memory
RK	Rünge-Kutta
SIMD	Single-Instruction Multiple-Data
SIMT	Single-Instruction Multiple-Thread
SISD	Single-Instruction Single-Data
SM	Streaming Multiprocessor
SP	Single Precision

A.2 List of Symbols

Table A.2: List of Symbols

Symbol	**Definition**
	A
$\mathbf{A}$	System matrix of a deterministic system in state space
$\mathbf{A}$	Matrix of coefficients in Butcher tableau in Chapter 2
	B
b	Width of a beam
$\mathbf{b}$	Vector of coefficients in Butcher tableau in Chapter 2
$\mathbf{B}$	Load distribution matrix of a deterministic system in configuration space
$\underline{\mathbf{B}}$	Load distribution matrix of a deterministic system in state space
	C

Continued on next page

Table A.2 – continued from previous page

Symbol	Definition
C_b	Computational cost of a brute-force method
C_p	Computational cost of a proposed method
$\mathbf{c}$	Vector of coefficients in Butcher tableau in Chapter 2
$\mathbf{C}$, $\mathbf{C}_i$, $\underline{\mathbf{C}}_j$	Output matrix of a deterministic system ($i = 0, 1, 2; j = 0, 1$)
$\mathbb{C}$	Set of complex numbers
D	
d	Depth of a beam
$\mathbf{D}$	Damping matrix of a deterministic system
$\mathbf{D}_u$	Damping matrix of a stochastic system
$\Delta\mathbf{D}$, $\delta\mathbf{D}$	Perturbation in the damping matrix of a system
E	
E	Young's modulus of elasticity
$\mathbb{E}[.]$	Eigenvalue operator
F	
$\mathbf{f}$, $\mathbf{F}$	Input to a computational model in configuration space
$\underline{\mathbf{f}}$, $\underline{\mathbf{F}}$	Input to a computational model in state space
G	
G_i	Gain in computational efficiency
$\mathbf{g}$	Response of a system to initial conditions in configuration space
$\underline{\mathbf{g}}$, $\widetilde{\underline{\mathbf{g}}}$	Impulse response function of a stochastic system in state space
H	
$\mathbf{h}$	Impulse response function of a system in configuration space
$\underline{\mathbf{h}}$	Impulse response function of a deterministic system in state space
$\mathbf{h}_L$	Impulse response function of a system in the pattern of uncertainty
$\mathbf{H}$	Transfer function matrix of a deterministic system in configuration space
$\mathbf{H}_u$, $\widetilde{\mathbf{H}}$	Transfer function matrix of a stochastic system in configuration space
$\underline{\mathbf{H}}$	Transfer function matrix of a system in state space
K	
$\mathbf{k}_i$	Rünge-Kutta stages in Chapter 2
$\mathbf{K}$	Stiffness matrix of a deterministic system
$\mathbf{K}_u$	Stiffness matrix of a stochastic system
$\Delta\mathbf{K}$, $\delta\mathbf{K}$	Perturbation in the stiffness matrix of a system
L	
L	Length of a beam
$\mathscr{L}[.]$	Laplace transform operator

Continued on next page

Table A.2 – continued from previous page

Symbol	Definition
$\mathbf{L}$, $\mathbf{L}_1$, $\widehat{\mathbf{L}}$	Influence matrix to map system uncertainties/nonlinearities to appropriate degrees-of-freedom
$\mathbf{L}_d$	Influence matrix for damping uncertainties
$\mathbf{L}_k$	Influence matrix for stiffness uncertainties
M	
$\mathbf{M}$	Mass matrix of a deterministic system
N	
n	Total degrees-of-freedom of a system in configuration space
$\underline{n}$	Total degrees-of-freedom of a system in state space
N_a	Number of quadrature points used by a numerical quadrature scheme
N_b	Number of Krylov block used for dimension reduction
N_f	Number of inputs to a system
N_{freq}	Number of discretization points in frequency domain
N_k	Size of the reduced system of nonlinear Volterra integral equations
N_o	Number of outputs desired
N_p	Number of uncertain/perturbed degrees-of-freedom of a system
N_{p_k}	Number of uncertain stiffness parameters of a system
N_{p_d}	Number of uncertain damping parameters of a system
N_{proc}	Number of parallel processes available for computations
N_r	Size of the reduced dimensional space
N_s	Total number of samples computed for a sampling-based strategy
N_{s_p}	Number of samples corresponding to parametric uncertainties
N_{s_f}	Number of samples corresponding to load uncertainties
N_t	Number of discretization points in time domain
P	
$\mathbf{p}_u$	Parameters of a system in uncertain relationships
$\mathbf{p}_n$	Parameters of a system in nonlinear relationships
$\mathbf{p}$	Pseudo force in time domain
$\mathbf{P}$	Pseudo force in frequency domain
Q	
$\mathbf{q}$	Modal displacement response of a system in configuration space
q_i	i^{th} modal coordinate in configuration space
$\underline{\mathbf{q}}$	Modal displacement response of a system in state space
$\underline{q}_i$	i^{th} modal coordinate in state space
$\mathbf{Q}$	Matrix whose columns are the elements of a generalized coordinate space

Continued on next page

Table A.2 – continued from previous page

Symbol	Definition
	R
R_{xx}	Autocorrelation function of x
$\mathbf{r}$	Residual vector
$\mathbb{R}$	Set of real numbers
	S
s	Frequency domain parameter
S_{xx}, $\mathbf{S_{xx}}$	Power spectral density of $\mathbf{x}$
S_{MC}	Gain in computational efficiency obtained by a multi-core CPU implementation with respect to a single-core CPU implementation
S_{GS}	Gain in computational efficiency obtained by a GPU implementation with respect to a single-core CPU implementation
S_{GM}	Gain in computational efficiency obtained by a GPU implementation with respect to a multi-core CPU implementation
$\mathcal{S}$	Conceptual representation of an input/output model
	T
t_0	Initial time
t	Time domain parameter
t_{sc}	Computational time of a sequential algorithm
t_p	Computational time of a parallel algorithm
t_c	Communication time of a parallel algorithm
T_i	i^{th} parallel thread
$\mathbf{T}$	Transformation matrix
Δt	Time-step
	U
$\mathbf{u}$	Vector function that prescribes uncertainties and nonlinearities of a system in configuration space
$\underline{\mathbf{u}}$	Vector function that prescribes uncertainties and nonlinearities of a system in state space
	V
$\mathbf{v}$, $\widehat{\mathbf{v}}$	Time domain displacement response of a stochastic system
$\mathbf{V}$	Frequency domain displacement response of a stochastic system
	W
$w_{i,j}$	Weights associated with a numerical quadrature rule
	X

Continued on next page

Table A.2 – continued from previous page

Symbol	Definition
$\mathbf{x}$, $\widehat{\mathbf{x}}$	Time-domain displacement response of a deterministic system in configuration-space
$\mathbf{x}_0$, $\dot{\mathbf{x}}_0$	Initial displacement and velocity of a deterministic system in configuration-space
$\underline{\mathbf{x}}$, $\underline{\widehat{\mathbf{x}}}$	Time-domain displacement response of a deterministic system in state-space
$\underline{\mathbf{x}}_0$	Initial displacement of a deterministic system in state-space
Y	
$\mathbf{y}$	Output of a computational model
Z	
Greek Symbols	
α, β	Coefficients of Rayleigh damping
$\mathbf{\Gamma}_{\mathbf{x}x}$	Covariance matrix of $\mathbf{x}$
$\delta(t)$	Dirac's delta function
ζ_i	i^{th} damping coefficient of a system
θ	Angle between exact and approximate eigenvectors
$\imath$	Square root of negative unity
$\mathbf{\Lambda}$	Diagonal matrix of eigenvalues of a deterministic system in configuration space
λ_i	i^{th} eigenvalue of a deterministic system in configuration space
$\underline{\mathbf{\Lambda}}$	Diagonal matrix of eigenvalues in state space
$\underline{\lambda}_i$	i^{th} eigenvalue in state space
$\boldsymbol{\mu}_{\mathbf{x}}$	Mean vector of $\mathbf{x}$
μ_x	Mean of x
ρ	Mass density of a beam material
$\mathbf{\Sigma}$	Diagonal matrix of eigenvalues of a stochastic system in configuration space
σ_i	i^{th} eigenvalue of a stochastic system in configuration space
τ	A time domain parameter
$\mathbf{\Phi}$, $\widehat{\mathbf{\Phi}}$	Modal matrix of mass-normalized eigenvectors of a deterministic system in configuration space
$\boldsymbol{\phi}_i$	i^{th} mass-normalized eigenvector of a deterministic system in configuration space
$\underline{\mathbf{\Phi}}$	Modal matrix of eigenvectors in state space
$\underline{\boldsymbol{\phi}}_i$	i^{th} eigenvector in state space

Continued on next page

Table A.2 – continued from previous page

Symbol	Definition
$\boldsymbol{\Psi}$, $\widehat{\boldsymbol{\Psi}}$	Modal matrix of mass-normalized eigenvectors of a stochastic system in configuration space
$\boldsymbol{\psi}_i$	i^{th} mass-normalized eigenvector of a stochastic system in configuration space
ω_{n_i}	i^{th} natural frequency of a system
ω_{d_i}	i^{th} damped-natural frequency of an under-damped system
$\omega^*_{d_i}$	i^{th} damped-natural frequency of an over-damped system
Others	
$\mathbf{0}_{m\times n}$	Matrix of zeros of size $m\times n$
$\mathbf{I}_{m\times n}$	Identity matrix of size $m\times n$
$\mathbf{1}$	Vector of all ones
$\forall$	For all
$\exists$	There exists