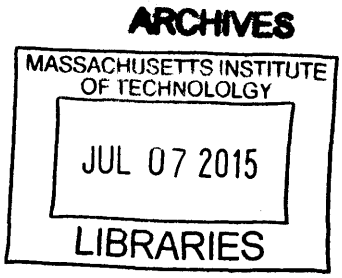
Uncertainty Quantification for Integrated Circuits and Microelectromechanical Systems



by

Zheng Zhang

M.Phil., The University of Hong Kong (2010)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

Signature redacted

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2015

Signature redacted

Certified by .....
Luca Daniel
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Accepted by .....
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

Uncertainty Quantification for Integrated Circuits and Microelectromechanical Systems

by

Zheng Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Uncertainty quantification has become an important task and an emerging topic in many engineering fields. Uncertainties can be caused by many factors, including inaccurate component models, the stochastic nature of some design parameters, external environmental fluctuations (e.g., temperature variation), measurement noise, and so forth. In order to enable robust engineering design and optimal decision making, efficient stochastic solvers are highly desired to quantify the effects of uncertainties on the performance of complex engineering designs.

Process variations have become increasingly important in the semiconductor industry due to the shrinking of micro- and nano-scale devices. Such uncertainties have led to remarkable performance variations at both circuit and system levels, and they cannot be ignored any more in the design of nano-scale integrated circuits and microelectromechanical systems (MEMS). In order to simulate the resulting stochastic behaviors, Monte Carlo techniques have been employed in SPICE-like simulators for decades, and they still remain the mainstream techniques in this community. Despite of their ease of implementation, Monte Carlo simulators are often too time-consuming due to the huge number of repeated simulations.

This thesis reports the development of several stochastic spectral methods to accelerate the uncertainty quantification of integrated circuits and MEMS. Stochastic spectral methods have emerged as a promising alternative to Monte Carlo in many engineering applications, but their performance may degrade significantly as the parameter dimensionality increases. In this work, we develop several efficient stochastic simulation algorithms for various integrated circuits and MEMS designs, including problems with both low-dimensional and high-dimensional random parameters, as well as complex systems with hierarchical design structures.

The first part of this thesis reports a novel stochastic-testing circuit/MEMS simulator as well as its advanced simulation engine for radio-frequency (RF) circuits. The proposed stochastic testing can be regarded as a hybrid variant of stochastic Galerkin and stochastic collocation: it is an intrusive simulator with decoupled computation and adaptive time stepping inside the solver. As a result, our simulator gains remark-

able speedup over standard stochastic spectral methods and Monte Carlo in the DC, transient and AC simulation of various analog, digital and RF integrated circuits. An advanced uncertainty quantification algorithm for the periodic steady states (or limit cycles) of analog/RF circuits is further developed by combining stochastic testing and shooting Newton. Our simulator is verified by various integrated circuits, showing $10^2\times$ to $10^3\times$ speedup over Monte Carlo when a similar level of accuracy is required.

The second part of this thesis presents two approaches for hierarchical uncertainty quantification. In hierarchical uncertainty quantification, we propose to employ stochastic spectral methods at different design hierarchies to simulate efficiently complex systems. The key idea is to ignore the multiple random parameters inside each subsystem and to treat each subsystem as a single random parameter. The main difficulty is to recompute the basis functions and quadrature rules that are required for the high-level uncertainty quantification, since the density function of an obtained low-level surrogate model is generally unknown. In order to address this issue, the first proposed algorithm computes new basis functions and quadrature points in the low-level (and typically high-dimensional) parameter space. This approach is very accurate; however it may suffer from the curse of dimensionality. In order to handle high-dimensional problems, a sparse stochastic testing simulator based on analysis of variance (ANOVA) is developed to accelerate the low-level simulation. At the high-level, a fast algorithm based on tensor decompositions is proposed to compute the basis functions and Gauss quadrature points. Our algorithm is verified by some MEMS/IC co-design examples with both low-dimensional and high-dimensional (up to 184) random parameters, showing about $10^2\times$ speedup over the state-of-the-art techniques. The second proposed hierarchical uncertainty quantification technique instead constructs a density function for each subsystem by some monotonic interpolation schemes. This approach is capable of handling general low-level possibly non-smooth surrogate models, and it allows computing new basis functions and quadrature points in an analytical way.

The computational techniques developed in this thesis are based on stochastic differential algebraic equations, but the results can also be applied to many other engineering problems (e.g., silicon photonics, heat transfer problems, fluid dynamics, electromagnetics and power systems).

There exist lots of research opportunities in this direction. Important open problems include how to solve high-dimensional problems (by both deterministic and randomized algorithms), how to deal with discontinuous response surfaces, how to handle correlated non-Gaussian random variables, how to couple noise and random parameters in uncertainty quantification, how to deal with correlated and time-dependent subsystems in hierarchical uncertainty quantification, and so forth.

Thesis Supervisor: Luca Daniel
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I am so happy to have had Prof. Luca Daniel as my thesis supervisor during this wonderful journey. Besides his patient guidance on my research and his consistent support to my career, Luca has entirely reshaped my research vision: he has taught me to be an interdisciplinary researcher. Thanks to his support, I have had the opportunities of interacting with many world-leading experts in various research fields (e.g., Prof. Boning working on semiconductor process modeling, Prof. Turitsyn working on power systems, Prof. Adalsteinsson, Prof. Riccardo Lattanzi and Prof. Sodickson working on MRI, Prof. Karniadakis and Prof. Marzouk working on uncertainty quantification). Luca also supported me to visit many other universities and institutes both in United States and in Europe. These visits have greatly enriched my research and cultural experience. Luca is a great professor and also a great friend.

My thesis committee members Prof. Jacob White, Prof. George Karniadakis and Prof. Duane Boning have provided extensive help and guidance in my PhD research. I learnt many new stuffs from every meeting with Jacob (ranging from MEMS, oscillators, model-order reduction to integral equation). Prof. Boning has showed me a wonderful top-level picture of process variation and statistical modeling, leading to lots of interesting ideas about inverse problems and robust design. I am so fortunate to have visited Prof. Karniadakis's group for a summer. The research collaboration with Prof. Karniadakis and his students (especially Dr. Xiu Yang, Dr. Zhongqiang Zhang and Dr. Heyrim Cho) has resulted in several published and upcoming papers, as well as many new ideas about uncertainty quantification.

I would like to sincerely thank my research collaborators (and mentors). Dr. Tarek Moselhy taught me many stuffs about uncertainty quantification when I entered this fantastic field. Dr. Mattan Kamon is a legend in parasitic extraction, and I learnt everything about MEMS simulation when I did my summer intern under his mentorship. Prof. Ibrahim Elfadel and Prof. Paolo Maffezzoni are my long-term collaborators on circuit simulation and uncertainty quantification. The collaboration with Dr. Xiu Yang have lead to several publications and ideas on uncertainty quan-

tification. Prof. Ivan Oseledets is a world-class expert working on tensors, and the interactions with him have resulted in many interesting solutions to high-dimensional function and data approximation.

The time spent with my smart and nice labmates is truly amazing. The research collaborations with Dr. Niloofar Farnoosh and Tsui-Wei (Lily) Weng are productive, and we also explored many resturants and tried lots of great food together. Dr. Yu-Chung Hsiao and Dr. Zohaib Mahmood are experts teaching me optimization and system identification. Richard Zhang is a great guy that can always provide inspiring ideas and insightful technical comments. Jose Cruz Serralles, Dr. Jorge Fernandez Villena and Dr. Athanasios G. Polimeridis are the best guys to discuss about MRI and integral equations. My former officemate Yan Zhao helped me a lot both in my study and in my daily life when I came to MIT. Thank you all, my great friends!

Finally, I would like to thank my wife for her accompanying me in cold Boston, for her consistent support through these years, and for her bearing some of my bad habits (such as sleeping very late).

I own this thesis to my beloved families in China.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Research Motivation

1.1 Process Variations in Nanoscale Design

Numerical simulation has been accepted as a standard step to verify the performance of integrated circuits and microelectromechanical systems (MEMS). By running a numerical simulator (e.g., SPICE and its variants [5–7] for circuit simulation, and Coventorware and MEMS+ [8,9] for MEMS simulation), the performance of a design case can be predicted and improved before the costly (and typically iterative) fabrication processes. In traditional semiconductor design, given a specific design strategy (e.g., the schematic and parameter values of a circuit or the 3-D geometry of a MEMS device) designers can run a simulator to predict the corresponding performance output (e.g., frequency of an oscillator circuit). In this type of analysis, it is assumed that no uncertainties influence chip performance. However, this is not true in today's semiconductor design.

Many nano-scale fabrication steps (such as lithography, chemical polishing, etc.) are subject to manufacturing variability. Consequently, randomness appears in the geometric and material properties of devices. As a demonstration, Fig. 1-1 shows some variations observed in practical circuit and MEMS fabrications. Such process variations can significantly degrade the performance of a circuit or MEMS device. For

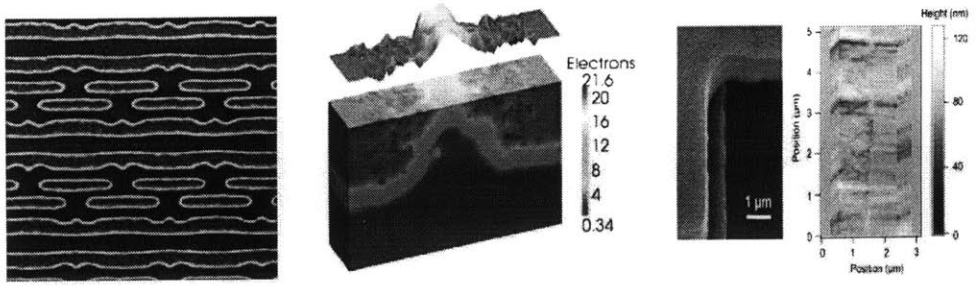


Figure 1-1: Left: geometric variations in 45-nm SRAM cells (courtesy of IBM). Middle: fluctuations of electron density in a 45-nm transistor (courtesy of Gold Standard Simulations Ltd.). Right: geometric imperfection in MEMS fabrications.

instance, the variations of transistor threshold voltage can lead to a huge amount of leakage power [10, 11] and thus becomes a severe problem in low-power design. The material and geometric uncertainties of VLSI interconnects [12–19] may cause huge timing variations [20, 21]. Device-level uncertainties can propagate to a circuit level and further influence a system-level behavior. Assume that we have have fabricated 1000 products for a given 3-D structure of a MEMS resonator. These MEMS chips may have different resonant frequencies due to the fabrication process variations. If we further utilize these MEMS resonators to build phase-lock loops (PLL), it is not suprising to find that some of PLLs cannot work properly due to such variations in MEMS resonators.

In this thesis we investigate uncertainty quantification techniques for integrated circuits and MEMS, hoping to characterize the effect of process variations on chip design and to improve design quality.

1.2 Uncertainties in Numerical Simulation

Many types of uncertainties may need to be considered in a general numerical modeling and simulation framework. Below we summarize a few uncertainty sources.

- **Parametric uncertainty**, normally named "**process variation**" in electronic design automation, may be present in the input variables of the computational

model. For example, the threshold voltages of different transistors on a single silicon wafer can be different due to the random doping effect. Consequently, different chips on the same wafer can have different performances.

- **Model uncertainty**, also called structural uncertainty or model inadequacy, is due to the inaccurate mathematical description of the physical or engineering problem. In fast circuit analysis the parasitic effects are sometimes ignored; in semiconductor device modeling, lots of simplifications and approximations are employed even in the most advanced models. Such approximation can lead to discrepancy between the simulated and actual results.

- **Parameter uncertainty** is due to the inaccuracy of some deterministic model parameters. Note that parameter uncertainties are differerent from "parametric uncertainties", since the latter are random variables. Assume that we have a voltage-dependent nonlinear resistor $R = a + bV$ where V is the voltage across the resistor. In practice, we may not known the exact values of the deterministic variables a and b, and thus some empirical values may be used in practical computational models.

- **Numerical uncertainty**. This is typically generated by the numerical errors and approximation. For example, when we solve a nonlinear equation $F(x) = 0$ by Newton's iteration (which is employed in almost all types of nonlinear circuit analysis), we may terminate the iteration and regard x_k as an accurate result if $||F(x_k)||_2 \leq \epsilon$. Actually, x_k is not exactly equal to x.

- **Experimental uncertainty**. When measuring the "actual" performance of a circuit or MEMS chip, some measurement errors will be inevitably introduced. For example, let the actual resonant frequency of a MEMS resonator be f_0, the practical measurement result is $f = f_0 + f_\epsilon$ where f_ϵ is a noise term.

In this thesis, we consider parametric uncertainties (i.e., process variations) only. This treatment is acceptable in most of design cases due to the maturity of device modeling and deterministic circuit/MEMS simulation techniques. Extensive

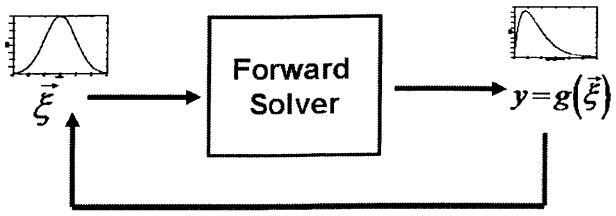


Figure 1-2: Forward and inverse problems.

literature has discussed how to extract the statistical information of process variations from experimental data [22–28]. Here we simply assume that a good description (e.g., probability density function) has been provided.

1.3 Forward and Inverse Problems

Assume that the process variations are represented by a set of random parameters $\vec{\xi} \in \Omega \subseteq \mathbb{R}^d$, and they are related to an output of interest $y(\vec{\xi})$ by a given computational model. Two kinds of problems may be of interest (c.f. Fig. 1-2).

1. **Forward Problems**. In a forward problem, descriptions of $\vec{\xi}$ are given, and one aims to estimate the statistical behavior of $y(\vec{\xi})$) by running a circuit or MEMS simulator. Such uncertainty quantification problems require specialized stochastic solvers to complete the numerical computation very efficiently.

2. **Inverse Problems**. In this task, one aims to estimate $\vec{\xi}$ given some measurement data of $y(\vec{\xi})$. This task requires iteratively solving many forward problems, and sometimes it is ill-posed.

In this thesis we will focus on developing fast solvers for forward problems. Based on the developed fast forward solvers, we hope to advance the techniques of solving inverse problems in the near future.

1.4 Computational Models in This Thesis

Assume that $\vec{\xi}$ is related to the output of interest y by a computational model:

$$\mathbb{F}\left(\vec{x}(t,\vec{\xi}),\vec{\xi}\right)=0,\ \ y\left(\vec{\xi}\right)=\mathbb{Y}\left(\vec{x}\left(t,\vec{\xi}\right)\right). \tag{1.1}$$

Here $\vec{x}(t,\vec{\xi}) \in \mathbb{R}^n$ are unknown variables (or state variables). Mapping from $\vec{x}(t,\vec{\xi})$ to the quantity of interest y by operator $\mathbb{Y}$ needs little extra computational cost.

Assumption 1. *throughout this thesis, we assume that all random parameters are independent, i.e., their joint probability density function (PDF) can be expressed as*

$$\rho(\vec{\xi})=\prod_{k=1}^{d}\rho_k\left(\xi_k\right), \tag{1.2}$$

with $\rho_k\left(\xi_k\right)$ being the PDF of $\xi_k \in \Omega_k$.

In (1.1), $\mathbb{F}$ is a general mathematical abstraction. In electromagnetic computation, $\mathbb{F}$ can be a Maxwell equation with uncertainties, which is actually a stochastic partial differential equation or a stochastic integral equation. In network-based integrated circuit and MEMS analysis, $\mathbb{F}$ can be a stochastic differential algebraic equation that describes the dynamics of state variables $\vec{x}(t,\vec{\xi})$, as is described below.

1.4.1 Stochastic Circuit Equation

In stochastic circuit simulation, modified nodal analysis (MNA) [29] can be utilized to obtain a stochastic differential algebraic equation

$$\mathbb{F}\left(\vec{x}(t,\vec{\xi}),\vec{\xi}\right)=0\ \Leftrightarrow\ \frac{d\vec{q}\left(\vec{x}\left(t,\vec{\xi}\right),\vec{\xi}\right)}{dt}+\vec{f}\left(\vec{x}\left(t,\vec{\xi}\right),\vec{u}(t),\vec{\xi}\right)=0 \tag{1.3}$$

where $\vec{u}(t) \in \mathbb{R}^m$ is the input signal (e.g., constant or time-varying current and voltage sources), $\vec{x} \in \mathbb{R}^n$ denotes nodal voltages and branch currents, $\vec{q} \in \mathbb{R}^n$ and $\vec{f} \in \mathbb{R}^n$ represent the charge/flux term and current/voltage term, respectively. Vector

$\vec{\xi} = [\xi_1; \xi_2; \cdots \xi_d]$ denotes d random variables describing the device-level uncertainties assumed mutually independent.

1.4.2 Stochastic MEMS Equation.

A MEMS design with process variations can be described by a 2nd-order differential equation

$$M\left(\vec{z}(t,\vec{\xi}),\vec{\xi}\right)\frac{d^2\vec{z}(t,\vec{\xi})}{dt^2} + D\left(\vec{z}(t,\vec{\xi}),\vec{\xi}\right)\frac{d\vec{z}(t,\vec{\xi})}{dt} + \tilde{f}\left(\vec{z}(t,\vec{\xi}),\vec{u}(t),\vec{\xi}\right) = 0 \qquad (1.4)$$

where $\vec{z} \in \mathbb{R}^{\tilde{n}}$ denotes displacements and rotations; $\vec{u}(t)$ denotes the inputs such as voltage sources or mechanical forces; $M,\ D \in \mathbb{R}^{\tilde{n}\times\tilde{n}}$ are the mass matrix and damping coefficient matrix, respectively; $\tilde{f}$ denotes the net forces from electrostatic and mechanical forces. This differential equation can be obtained by discretizing a partial differential equation or an integral equation [30], or by using the fast hybrid platform that combines finite-element/boundary-element models with analytical MEMS device models [8,31–33]. This 2nd-order differential equation can be easily converted to the form of (1.3) by letting $n = 2\tilde{n}$

$$\vec{x} = \begin{pmatrix} \vec{z} \\ \frac{d\vec{z}}{dt} \end{pmatrix}, \quad \vec{f} = \begin{pmatrix} \tilde{f} \\ 0 \end{pmatrix}, \quad \frac{d\vec{q}}{dt} = \begin{pmatrix} D & M \\ & I \end{pmatrix}\frac{d\vec{x}}{dt} \qquad (1.5)$$

In this expression we have omitted the variables that influence each term.

1.5 Thesis Contribution and Organization

1.5.1 Contributions of This Thesis

This thesis focuses on the development of efficient forward solvers to accelerate the uncertainty quantification of integrated circuits and MEMS. The novel contributions include two parts.

In the first part, we propose novel intrusive algorithms to compute the uncertain-

ties propogating from devices to circuits.

- **Contribution 1.** In Chapter 4, we propose an efficient intrusive solver called stochastic testing [34, 35]. This formulation can be regarded as a hybrid version of stochastic collocation and stochastic Galerkin. On one side, similar to stochastic collocation its complexity is linearly dependent on the number of basis functions since decoupling can be exploited inside a Newton's iteration. On the other hand, it sets up a coupled deterministic differential equation by using only a small portion of quadrature points, such that adaptive time stepping can be implemented to accelerate the computation of coefficients in the generalized polynomial-chaos expansion of $\vec{x}(\vec{\xi}, t)$.

- **Contribution 2.** Based on the proposed stochastic testing circuit simulator, an advanced numerical solver is presented in Chapter 5 to quantify the uncertainty of periodic steady states that are frequently used in analog/RF circuit and power electronic circuit simulations [36]. By combining stochastic testing with Newton's shooting, novel periodic steady-state solvers for both forced circuits and oscillator circuits are developed.

The second part of this thesis develops computational techniques that estimate the system-level uncertainties induced by fabrication process variations.

- **Contribution 3.** Chapter 6 proposes a high-dimensional hierarchical algorithm that employs stochastic spectral methods at different levels of design hierarchy to simulate a complex system [37]. When the parameter dimensionality is high, it is too expensive to extract a surrogate model for each subsystem by using any standard stochastic spectral method. Furthermore, it is also non-trivial to perform high-level simulation with a stochastic spectral method, due to the high-dimensional integration involved when computing the basis functions and Gauss quadrature rules for each subsystem. In order to reduce the computational cost, some fast numerical algorithms are developed to accelerate the simulations at both levels [37, 38].

- **Contribution 4.** Chapter 7 proposes an alternative approach to enable hierarchical uncertainty quantification [39]. In order to handle general stochastic surrogate models that may be non-smooth, we propose to compute the basis functions and quadrature points/weights by first approximating the underlying density functions. In Chapter 7, we tackle this problem by two monotonic density estimation techniques.

1.5.2 Thesis Outline

This thesis is organized as follows:

- In Chapter 2, we give an introduction to some mathematical background. We aim to make this background chapter as brief as possible.

- Chapter 3 surveys different computational techniques for solving forward uncertainty quantification problems. We also review their applications in previous circuit and MEMS simulation.

- Chapter 4 to Chapter 7 present the details of our four novel contributions, including how to quantify the uncertainties propogating from devices to circuits and how to compute them efficiently in a hierarchical complex system. In these chapters, we will demonstrate various application examples arising from integrated circuits and MEMS design.

- Finally, Chapter 8 summarizes the results of this thesis and discusses some future work in this field.

Chapter 2

Mathematical Background

This chapter introduces some background about generalized polynomial chaos, numerical integration and tensors that will be used in the subsequent chapters.

2.1 Generalized Polynomial Chaos Expansion

Generalized polynomial chaos was introduced by Xiu and Karniadakis in 2002 [40], and it has been widely used in stochastic spectral methods [4,41–44]. As a generalization of Hermite-type polynomial chaos expansion [45] that approximates $\vec{x}(t,\vec{\xi})$ with Gaussian random parameters, a generalized polynomial chaos expansion can handle both Gaussian and non-Gaussian random parameters efficiently.

If $\vec{x}(\vec{\xi},t)$ is smooth enough and has a bounded 2nd-order moment, it can be approximated by a finite-term generalized polynomial-chaos expansion

$$\vec{x}(t,\vec{\xi}) \approx \tilde{x}(t,\vec{\xi}) = \sum_{\vec{\alpha}\in\mathcal{P}} \hat{x}_{\vec{\alpha}}(t) H_{\vec{\alpha}}(\vec{\xi}) \tag{2.1}$$

where $H_{\vec{\alpha}}(\vec{\xi})$ is a basis function indexed by $\vec{\alpha}$, $\hat{x}_{\vec{\alpha}}(t) \in \mathbb{R}^n$ denotes the corresponding weight (or coefficient) for the basis function, and $\mathcal{P}$ is a set containing some properly selected index vectors.

Definition 1 (Inner Product). *In the stochastic space Ω and with a probability density function $\rho(\vec{\xi})$, the inner product of any two general functions $y_1(\vec{\xi})$ and $y_2(\vec{\xi})$*

is defined as

$$\left\langle y_1(\vec{\xi}), y_2(\vec{\xi}) \right\rangle_{\Omega,\rho(\vec{\xi})} = \int_{\Omega} \rho(\vec{\xi}) y_1(\vec{\xi}) y_2(\vec{\xi}) d\vec{\xi}. \tag{2.2}$$

In the generalized polynomial-chaos expansion (2.1), the basis functions are chosen in a special way such that they are orthonormal to each other

$$\left\langle H_{\vec{\alpha}}(\vec{\xi}), H_{\vec{\beta}}(\vec{\xi}) \right\rangle_{\Omega,\rho(\vec{\xi})} = \delta_{\vec{\alpha},\vec{\beta}}.$$

Here $\delta_{\vec{\alpha},\vec{\beta}}$ is a Delta function (the value of $\delta_{\vec{\alpha},\vec{\beta}}$ is 1 if $\vec{\alpha} = \vec{\beta}$, and 0 otherwise). The basis functions are computed according to the density function of each random parameter, as described below.

2.1.1 Constructing Basis Functions: Univariate Case

Consider a single random parameter $\xi_k \in \Omega_k \subseteq \mathbb{R}$. Given its marginal density function $\rho_k(\xi_k)$, one can construct a set of polynomial functions subject to the orthonormal condition:

$$\left\langle \phi_\nu^k(\xi_k), \phi_{\nu'}^k(\xi_k) \right\rangle_{\Omega_k,\rho_k(\xi_k)} = \int_{\Omega_k} \phi_\nu^k(\xi_k) \phi_{\nu'}^k(\xi_k) \rho_k(\xi_k) d\xi_k = \delta_{\nu,\nu'} \tag{2.3}$$

where $\langle , \rangle_{\Omega_k,\rho_k(\xi_k)}$ denotes the inner product in Ω_k with density function $\rho_k(\xi_k)$; $\delta_{\nu,\nu'}$ is a Delta function; integers ν and ν' are the highest degrees of ξ_k in polynomials $\phi_\nu^k(\xi_k)$ and $\phi_{\nu'}^k(\xi_k)$, respectively. In order to satisfy the constraint (2.3), one can construct polynomials $\{\phi_\nu^k(\xi_k)\}_{\nu=0}^p$ by the following procedures [46]:

1. construct a set of orthogonal polynomials $\{\pi_\nu^k(\xi_k)\}_{\nu=0}^p$ with an leading coefficient 1 according to the recurrence relation

$$\pi_{\nu+1}^k(\xi_k) = (\xi_k - \gamma_\nu)\, \pi_\nu^k(\xi_k) - \kappa_\nu \pi_{\nu-1}^k(\xi_k), \quad \nu = 0,\ 1, \cdots p-1$$

with initial conditions $\pi_{-1}^k(\xi_k) = 0$, $\pi_0^k(\xi_k) = 1$ and $\kappa_0 = 1$. For $\nu \geq 0$, the

Table 2.1: Univariate generalized polynomial-chaos (gPC) polynomial basis of some typical random parameters [4].

Distribution of ξ_k	PDF of ξ_k $[\rho_k(\xi_k)]$[1]	univariate gPC basis $\phi_\nu^k(\xi_k)$	Support Ω_k
Gaussian	$\frac{1}{\sqrt{2\pi}}\exp\left(\frac{-\xi_k^2}{2}\right)$	Hermite-chaos polynomial	$(-\infty,+\infty)$
Gamma	$\frac{\xi_k^{\gamma-1}\exp(-\xi_k)}{\Gamma(\gamma)}$, $\gamma>0$	Laguerre-chaos polynomial	$[0,+\infty)$
Beta	$\frac{{\xi_k}^{\alpha-1}(1-\xi_k)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}$, $\alpha,\beta>0$	Jacobi-chaos polynomial	$[0,1]$
Uniform	$\frac{1}{2}$	Legendre-chaos polynomial	$[-1,1]$

[1] $\Gamma(\gamma)=\int_0^\infty t^{\gamma-1}\exp(-t)\,dt$ and $\mathrm{B}(\alpha,\beta)=\int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\,dt$ are the Gamma and Beta functions, respectively.

recurrence parameters are defined as

$$\gamma_\nu=\frac{\mathbb{E}\left(\xi_k(\pi_\nu^k)^2(\xi_k)\right)}{\mathbb{E}\left((\pi_\nu^k)^2(\xi_k)\right)},\ \kappa_{\nu+1}=\frac{\mathbb{E}\left(\xi_k(\pi_{\nu+1}^k)^2(\xi_k)\right)}{\mathbb{E}\left(\xi_k(\pi_\nu^k)^2(\xi_k)\right)}. \tag{2.4}$$

Here $\mathbb{E}$ denotes the operator that calculates expectation.

2. obtain $\{\phi_\nu^k(\xi_k)\}_{\nu=0}^p$ by normalization:

$$\phi_\nu^k(\xi_k)=\frac{\pi_\nu^k(\xi_k)}{\sqrt{\kappa_0\kappa_1\cdots\kappa_\nu}},\ \text{for } \nu=0,1,\cdots,p. \tag{2.5}$$

Some univariate generalized polynomial-chaos basis functions are listed in Table 2.1 as a demonstration. It is worth noting that:

1. the univariate basis functions are not limited to the cases listed in Table 2.1;

2. when ξ_k is a Gaussian variable, its polynomial basis functions simplify to the Hermite polynomial chaos [45].

2.1.2 Constructing Basis Functions: Multivariate Cases.

When the components of $\vec{\xi}$ are assumed mutually independent, the multivariate generalized polynomial chaos can be constructed based on the univariate generalized polynomial chaos of each ξ_k. Given an index vector $\vec{\alpha}=[\alpha_1,\cdots,\alpha_d]\in\mathbb{N}^d$, the

corresponding multivariate generalized polynomial chaos is constructed as

$$H_{\vec{\alpha}}(\vec{\xi}) = \prod_{k=1}^{d} \phi_{\alpha_k}^{k}(\xi_k). \tag{2.6}$$

According to (2.3) and (1.2), it is straightforward to verify that obtained multivariate functions are orthonormal, i.e.,

$$\left\langle H_{\vec{\alpha}}(\vec{\xi}), H_{\vec{\beta}}(\vec{\xi}) \right\rangle_{\Omega,\rho(\vec{\xi})} = \int_{\Omega} H_{\vec{\alpha}}(\vec{\xi}) H_{\vec{\beta}}(\vec{\xi}) \rho(\vec{\xi}) d\vec{\xi} = \delta_{\vec{\alpha},\vec{\beta}}.$$

Note that $H_{\vec{\alpha}}(\vec{\xi})$ is the product of different types of univariate polynomials when ξ_k's have different density functions.

2.1.3 Selecting Index Set $\mathcal{P}$

An infinite number of basis functions may be required to obtain the exact value of $x(t, \vec{\xi})$. However, a finite number of basis functions can provide a highly accurate approximation in many engineering problems. Given $p \in \mathbb{N}^+$, there are two popular choices for $\mathcal{P}$ [43]:

1. **tensor product method**. In the tensor product method, one sets $\mathcal{P} = \{\vec{\alpha} | \ 0 \leq \alpha_k \leq p\}$, leading to totally $(p+1)^d$ generalized polynomial chaos bases in (2.1).

2. **total degree method**. In order to reduce the number of basis functions, the total degree scheme sets $\mathcal{P} = \{\vec{\alpha} | \ \alpha_k \in \mathbb{N}, \ 0 \leq \alpha_1 + \cdots + \alpha_d \leq p\}$, leading to

$$K = \begin{pmatrix} p+d \\ p \end{pmatrix} = \frac{(p+d)!}{p!d!} \tag{2.7}$$

basis functions in total.

There is a one-to-one correspondence between k (with $1 \leq k \leq K$) and the index vector $\vec{\alpha}$, thus for simplicity (2.1) is usually rewritten as

$$\vec{x}(t,\vec{\xi}) \approx \tilde{x}(t,\vec{\xi}) = \sum_{k=1}^{K} \hat{x}_k(t) H_k(\vec{\xi}). \tag{2.8}$$

2.1.4 Advantages of Generalized Polynomial-Chaos

The most prominent advantage of generalized polynomial chaos is its fast convergence rate: it converges exponentially for some analytical functions [4, 40, 44] as p increase. Strict exponential convergence rates may not be observed in practical engineering problems, but polynomial-chaos expansion still converge very fast (almost exponentially) when the function of interest has a smooth dependence on $\vec{\xi}$.

The second advantage is the convenience to extract some statistical information. Thanks to the orthonormality of $H_{\vec{\alpha}}(\vec{\xi})$'s in polynomial-chaos approximations, the mean value and standard deviation of $\vec{x}(\vec{\xi},t)$ are easily calculated as:

$$\mathbb{E}\left(\vec{x}\left(t,\vec{\xi}\right)\right) = \hat{x}_{\vec{\alpha}=0}(t), \text{ and } \sigma\left(\vec{x}\left(t,\vec{\xi}\right)\right) = \sqrt{\sum_{\vec{\alpha}\neq 0} |\hat{x}_{\vec{\alpha}}(t)|^2}. \tag{2.9}$$

Here $\sigma()$ means standard deviation. High-order moments may also be calculated in an analytical or numerical way.

2.1.5 Extension to Correlated Cases

Let $\rho_k(\xi_k)$ be the marginal density function of $\xi_k \in \Omega_k$ and $\rho(\vec{\xi})$ be the joint density function of $\vec{\xi} \in \Omega$. When the random parameters are correlated, orthonormal multivariate polynomial basis functions cannot be computed by applying (2.6). Given the orthonormal polynomials $\{\phi^k_{\alpha_k}(\xi_k)\}^p_{\alpha_k=0}$ for parameter ξ_k with marginal density function $\rho_k(\xi_k)$, one can construct a set of orthonormal basis functions by [47]

$$H_{\vec{\alpha}}(\vec{\xi}) = \sqrt{\frac{\rho_1(\xi_1)\cdots\rho_d(\xi_d)}{\rho(\vec{\xi})}} \prod_{k=1}^{d} \phi^k_{\alpha_k}(\xi_k). \tag{2.10}$$

The numerical implementation is not trivial since the joint density function must be marginalized. A stochastic-collocation implementation has been reported in [48] to simulate the uncertainties of silicon photonic devices with Gaussian-mixture process variations. Since the resulting basis functions are not polynomials any more, it is observed in [48] that many basis functions may be required to approximate a smooth output of interest.

In this thesis, it is assumed that all random parameters are mutually independent.

2.2 Numerical Integration

This section briefly reviews some popular numerical integration schemes that will be used later in some stochastic spectral methods.

2.2.1 Univariate Case

Given $\xi_k \in \Omega_k$ with a density function $\rho_k(\xi_k)$ and a function $g(\xi_k)$, one can employ a quadrature method to evaluate the integral

$$\int_{\Omega_k} g(\xi_k)\rho_k(\xi_k)d\xi_k \approx \sum_{j=1}^{\hat{n}} g(\xi_k^j)w_k^j. \tag{2.11}$$

The quadrature points ξ_k^j's and weights w_k^j's are chosen according to Ω_k and $\rho_k(\xi_k)$. There are two classes of quadrature rules: random and deterministic approaches.

Randomized Algorithms. Monte Carlo and its variants belong to the first class, which can be utilized regardless of the smoothness of $g(\xi_k)$. The basic idea is as follows. One first picks N samples according to the density function $\rho_k(\xi_k)$, then evaluates function $g(\xi_k)$ at each sample, and finally computes the integral as the average of all samples of $g(\xi_k)$. In Monte Carlo, the numerical error is proportional to $1/\sqrt{N}$, and thus a huge number of samples are required to achieve high accuracy.

When $g(\xi_k)$ is a smooth function, deterministic quadrature rules such as Gauss quadrature [49] and Clenshaw-Curtis rules [50, 51] can be employed. Such determin-

istic approaches can employ only a small number of samples to evaluate the integral with high accuracy.

Gauss Quadrature Method. With $\hat{n}$ points, Gauss quadrature rule produces exact results for all polynomials of degree $\leq 2\hat{n}-1$. Gauss quadrature rule is closely related to orthonormal basis functions as we described in Section 2.1.1. One can obtain $(m+1)$ Gauss quadrature points and weights $\{(\xi_k^j, w_k^j)\}_{j=1}^{m+1}$ by reusing the recurrence parameters obtained in (2.4). The parameters κ_ν^k's and γ_ν^k's are used to form a symmetric tridiagonal matrix $\mathbf{J} \in \mathbb{R}^{(m+1)\times(m+1)}$:

$$\mathbf{J}(\nu,\nu') = \begin{cases} \gamma_{\nu-1}^k, \text{ if } \nu = \nu' \\ \sqrt{\kappa_\nu^k}, \text{ if } \nu = \nu'+1 \\ \sqrt{\kappa_{\nu'}^k}, \text{ if } \nu = \nu'-1 \\ 0, \text{ otherwise} \end{cases} \quad \text{for } 1 \leq \nu, \nu' \leq m+1. \tag{2.12}$$

Let $\mathbf{J} = \mathbf{U}\Sigma\mathbf{U}^T$ be an eigenvalue decomposition, where $\mathbf{U}$ is a unitary matrix. The ν-th quadrature point and weight of ξ_k are $\Sigma(\nu,\nu)$ and $(\mathbf{U}(1,\nu))^2$, respectively [49].

Clenshw-Curtis Method. Using $\hat{n} = 2l$ quadrature points, Clenshaw-Curtis gets exact results when the degree of $g(\xi_k)$ is $\leq \hat{n}-1$. Clenshaw-Curtis scheme generates nested quadrature points and assumes that ξ_k is uniformly distributed in a bounded domain $[-1,1]$.

2.2.2 Multi-Dimensional Case

One can also evaluate a multidimensional integral in $\Omega \subseteq \mathbb{R}^d$ using the formula

$$\int_\Omega g(\vec{\xi})\rho(\vec{\xi})d\vec{\xi} \approx \sum_{j=1}^{\hat{N}} g(\vec{\xi}^j)w^j. \tag{2.13}$$

where $\hat{N}$ is the total number of quadrature points, and w^j is the weight corresponding to quadrature point $\vec{\xi}^j$.

Assume that for each random parameter ξ_k, $\hat{n}$ quadrature points and weights $\{\xi_k^j, w_k^j\}_{j=1}^{\hat{n}}$ have already been computed. Multidimensional quadrature rules can be

constructed in various ways based on the available one-dimensional quadrature rules.

Tensor-Product Method. With the obtained 1-D quadrature points for each ξ_k, a tensor-product rule generates $\hat{N} = \hat{n}^d$ pairs of multivariate quadrature points and weights to evaluate (2.13). Let $\vec{\alpha} = [\alpha_1, \cdots, \alpha_d] \in \mathbb{N}^d$ be a vector index, there is a one-to-one correspondence between $\vec{\alpha}$ and j for $1 \leq j \leq \hat{N}$. Then one can change the notations and denote the j-th quadrature point and weight by $\{\vec{\xi}^{\vec{\alpha}}, w^{\vec{\alpha}}\}$, and one has the corresponding multi-dimensional quadrature point and weight

$$\vec{\xi}^{\vec{\alpha}} = [\xi_1^{\alpha_1}, \cdots, \xi_d^{\alpha_d}], \text{ and } w^{\vec{\alpha}} = \prod_{k=1}^{d} w_k^{\alpha_k}. \tag{2.14}$$

Sparse Grids. With Smolyak's algorithm, sparse grid techniques [52,53] may use much fewer quadrature points than the tensor-product rule, thus they are also widely used to solve stochastic PDEs [54–57]. In [54–57] Smolyak's algorithm produces nested sparse grids because all random parameters are assumed uniformly distributed (and thus Clenshaw-Curtis rule is used for all ξ_k's). However, Smolyak's algorithm generates non-nested sparse grids when non-nested 1-D quadrature points are used for some parameters (since a random parameter with non-uniform distribution may not be handled effectively by the Clenshaw-Curtis rule). In sparse-grid approaches, the total number of quadrature points $\hat{N}$ is a polynomial function of d and $\hat{n}$.

Randomized Algorithms. When the 1-D quadrature points are independently generated by Monte Carlo, $\hat{N} = \hat{n}$ multi-dimensional quadrature points can be easily constructed, with the j-th point $\vec{\xi}^j = [\xi_1^j, \cdots, \xi_d^j]$. Again, the multidimensional integral in (2.13) can be evaluated as the average of $\{g(\vec{\xi}^j)\}_{j=1}^{\hat{N}}$.

2.2.3 Selection of Numerical Integration Schemes

A proper approach should be selected dependent on the practical integrand function $g(\vec{\xi})$.

When $g(\vec{\xi})$ is smooth and when the parameter dimensionality is not high, deterministic quadrature rules are good choices. When d is very small (i.e., below 5), tensor-product Gauss quadrature rules can work very well. As d becomes larger (e.g.,

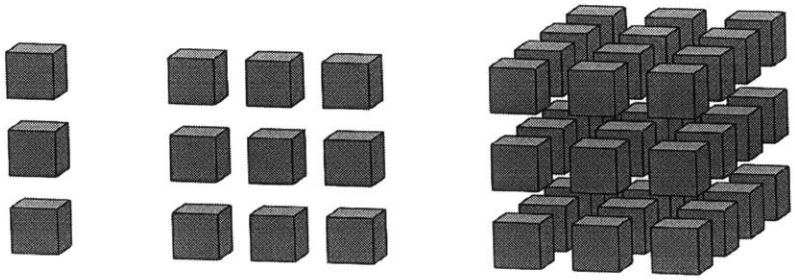


Figure 2-1: Demonstration of a vector (left), a matrix (center) and a 3-mode tensor (right).

below 30), sparse grids are better choices since much fewer quadrature points are required. When the parameter dimensionality is very high, Monte Carlo approaches are the only choice since $\hat{N}$ is independent of d (although thousands or millions of samples are still required). Quasi-Monte Carlo uses a deterministic sequence of samples to evaluate multi-dimensional integration [58]. It can improve the convergence rate of Monte Carlo when the parameter dimensionality is not too high, but it still has curse-of-dimensionality problems for very high-dimensional problems.

When $g(\vec{\xi})$ non-smooth (e.g., when it is an indicator function in failure probability analysis), Monte-Carlo-type algorithms are the only choice.

2.3 Tensors

2.3.1 Concepts Related to Tensors

Definition 2 (Tensor). *A tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_d}$ is a multi-mode (or multi-way) data array. The mode (or way) is d, which is also the total number of dimensions. The size of the k-th dimension is N_k. An element of the tensor is $\boldsymbol{\mathcal{A}}(i_1, \cdots, i_d)$, where the positive integer i_k is the index for the k-th dimension and $1 \leq i_k \leq N_k$. The total number of elements of $\boldsymbol{\mathcal{A}}$ is $N_1 \times \cdots \times N_d$.*

As a demonstration, Fig. 2-1 shows a vector (1-mode tensor) in $\mathbb{R}^{3\times 1}$, a matrix (2-mode tensor) in $\mathbb{R}^{3\times 3}$ and a 3-mode tensor in $\mathbb{R}^{3\times 3\times 4}$ in, where each small cube

represents a scalar.

Definition 3 (Inner Product of Two Tensors). *For $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_d}$, their inner product is defined as the sum of their element-wise product*

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1, \cdots, i_d} \mathcal{A}(i_1, \cdots i_d)\, \mathcal{B}(i_1, \cdots i_d). \tag{2.15}$$

Definition 4 (Frobenius Norm of A Tensor). *For $\mathcal{A} \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_d}$, its Frobenius norm is defined as*

$$\|\mathcal{A}\|_F = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}. \tag{2.16}$$

Definition 5 (Rank-One Tensors). *A d-mode tensor $\mathcal{A} \in \mathbb{R}^{N_1 \times \cdots \times N_d}$ is rank one if it can be written as the outer product of d vectors*

$$\mathcal{A} = \mathbf{v}^{(1)} \circ \mathbf{v}^{(2)} \cdots \circ \mathbf{v}^{(d)}, \text{ with } \mathbf{v}^{(k)} \in \mathbb{R}^{N_k} \tag{2.17}$$

where $\circ$ denotes the outer product operation. This means that

$$\mathcal{A}(i_1, \cdots, i_d) = \prod_{k=1}^{d} \mathbf{v}^{(k)}(i_k) \text{ for all } 1 \leq i_k \leq N_k. \tag{2.18}$$

Here $\mathbf{v}^{(k)}(i_k)$ denotes the i_k-th element of vector $\mathbf{v}^{(k)}$.

Definition 6 (Tensor Rank). *The rank of $\mathcal{A} \in \mathbb{R}^{N_1 \times \cdots \times N_d}$ is the smallest positive integer $\bar{r}$, such that*

$$\mathcal{A} = \sum_{j=1}^{\bar{r}} \mathbf{v}_j^{(1)} \circ \mathbf{v}_j^{(2)} \cdots \circ \mathbf{v}_j^{(d)}, \text{ with } \mathbf{v}_j^{(k)} \in \mathbb{R}^{N_k}. \tag{2.19}$$

2.3.2 Tensor Decomposition

It is attractive to perform tensor decomposition: given a small integer $r < \bar{r}$, approximate $\mathcal{A} \in \mathbb{R}^{N_1 \times \cdots \times N_d}$ by a rank-r tensor. Popular tensor decomposition algorithms include canonical decomposition [59–61] and Tuker decomposition [62, 63]. Canonical tensor decomposition aims to approximate $\mathcal{A}$ by the sum of r rank-1 tensors

[in the form of (2.19)] while minimizing the approximation error, which is normally implemented with alternating least square [60]. This decomposition scales well with the dimensionality d, but it is ill-posed for $d \geq 3$ [64]. Tucker decomposition aims to represent a tensor by a small core tensor and some matrix factors [62, 63]. This decomposition is based on singular value decomposition. It is robust, but the number of elements in the core tensor still grows exponentially with d.

Alternatively, tensor-train decomposition [65–67] approximates $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N_1 \times \cdots \times N_d}$ by a low-rank tensor $\hat{\boldsymbol{\mathcal{A}}}$ with

$$\hat{\boldsymbol{\mathcal{A}}}(i_1, \cdots i_d) = \boldsymbol{\mathcal{G}}_1(:, i_1, :)\, \boldsymbol{\mathcal{G}}_2(:, i_1, :) \cdots \boldsymbol{\mathcal{G}}_d(:, i_d, :)\,. \tag{2.20}$$

Here $\boldsymbol{\mathcal{G}}_k \in \mathbb{R}^{r_{k-1} \times N_k \times r_k}$, and $r_0 = r_d = 1$. By fixing the second index i_k, $\boldsymbol{\mathcal{G}}_k(:, i_k, :) \in \mathbb{R}^{r_{k-1} \times r_k}$ becomes a matrix (or vector when k equals 1 or d). To some extent, tensor-train decomposition have the advantages of both canonical tensor decomposition and Tuker decomposition: it is robust since each core tensor is obtained by a well-posed low-rank matrix decomposition [65–67]; it scales linearly with d since storing all core tensors requires only $O(Nr^2d)$ memory if we assume $N_k = N$ and $r_k = r$ for $k = 1, \cdots, d-1$. Given an error bound ϵ, the tensor train decomposition in (2.20) ensures

$$\left\| \boldsymbol{\mathcal{A}} - \hat{\boldsymbol{\mathcal{A}}} \right\|_F \leq \varepsilon \left\| \boldsymbol{\mathcal{A}} \right\|_F \tag{2.21}$$

while keeping r_k's as small as possible [66].

Definition 7 (TT-Rank). *In tensor-train decomposition (2.20) $\boldsymbol{\mathcal{G}}_k \in \mathbb{R}^{r_{k-1} \times N_k \times r_k}$ for $k = 1, \cdots d$. The vector $\vec{r} = [r_0, r_1, \cdots, r_d]$ is called TT-rank.*

Recently, tensor decomposition has shown promising applications in high-dimensional data compression [68–71] and in machine learning [72, 73]. Some efficient high-dimensional stochastic PDE solvers have been developed based on canonical tensor decomposition [74–76] (which is called "Proper Generalized Decomposition" in some papers) and tensor-train decomposition [77–80]. In [81], tensor-train decomposition was employed for high-dimensional function approximation.

Chapter 3

Survey of Forward Uncertainty Quantification Solvers

3.1 A Top-Level Figure

We would like to present a high-level engineering perspective of uncertainty quantification before discussing any specific numerical solver. In practice, users aim to complete various engineering tasks under some uncertainties (as shown in Fig. 3-1). Some typical examples include:

- **Yield or reliability analysis.** In many circuit design cases (e.g., in SRAM design), designers are interested in the probability that a chip fails to work. In the design of a MEMS switch, designers may be interested in the life time (i.e., the maximum number of turn-on/turn-off before the device fails to work). The estimated yield or reliability information can help designers to optimize their design.

- **Pricing or Binning.** The fabricated chips are subject to performance uncertainties due to process variations. A good decision on pricing could leverage information about the statistics of chip performance (e.g., speed and power consumption of the fabricated chips).

- **Hierarchical Design.** In order to build complex systems, designers usually

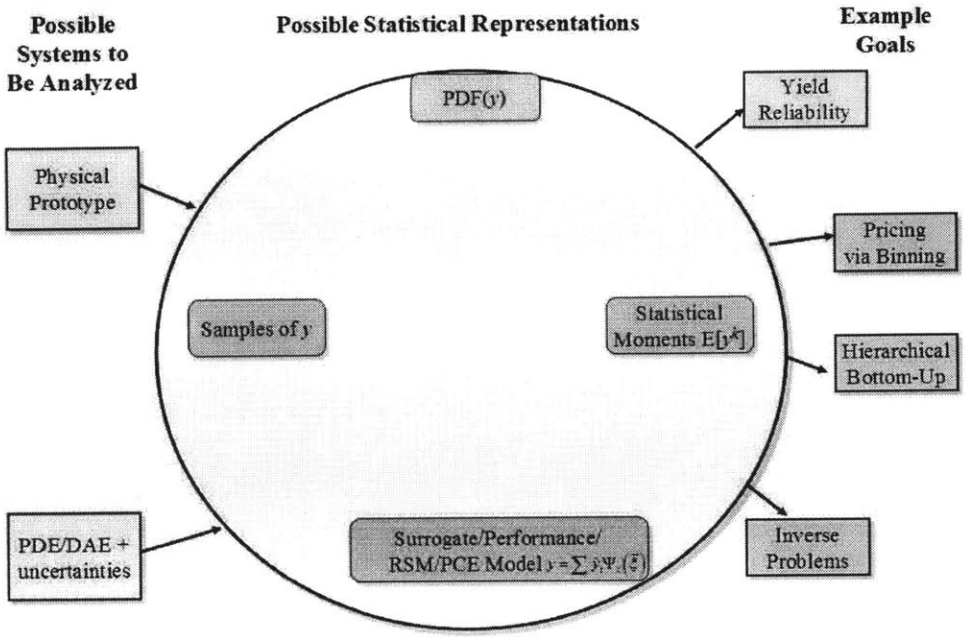


Figure 3-1: A top-level figure of uncertainty quantification (discussion with Prof. Luca Daniel and Dr. Tarek El-Moselhy).

employ hierarchical design methodology by reusing existing circuits or devices as some subsystems in a complex system. When uncertainties exist, the yield and reliability of the whole system may be estimated according to the statistical information of each subsystem.

- **Inverse Problems.** Very often, engineers have some measurement data of a circuit or system, and they want to infer the information of some parameters at the lower level. For instance, when modeling a new generation of semiconductor fabrication process, circuit designers may have the measurement data of some testing circuits. Using these data, it is possible to infer the distribution of some device parameters (e.g., transistor threshold voltage).

In order to complete a specific engineering task, a proper form can be chosen to represent an output of interest y. Popular representations include (but are not limited to) samples, density functions, statistical moments or surrogate models. In a general sense a surrogate model can be any approximation of y that can be easily

evaluated (for instance a reduced-order model). This thesis focuses on a specific form that represents y by some basis functions of $\vec{\xi}$. The representations in Fig. 3-1 are interchangeable under certain conditions. Here we ignore the algorithms that transform one representation to another since they are beyond the scope of this thesis.

A key problem in many practical uncertainty quantification problems is how to obtain a first representation in Fig. 3-1. This step depends on the available information. If physical prototypes (e.g., some fabricated chips) are available, one may be able to collect some measurement samples and then estimate the moments [82]. Since we aim to predict the uncertainties of a circuits or MEMS design before fabricating the chips, we instead start from a computational model. It is possible to obtain either kind of representation if a proper computational model is provided. For instance, starting from a response-excitation probability-density-function equation [83] one may be able to compute the joint density function of some quantities when noise is present in the input signals. Some numerical techniques are also available to compute some low-order moments from a stochastic computational model [84].

This chapter provides a brief survey of some popular techniques that firstly obtain some samples or a surrogate model from the computational model (1.1). These techniques can be classified into two classes: non-intrusive solvers and intrusive solvers.

3.2 Overview of Non-intrusive and Intrusive Solvers

Non-intrusive Solvers are also called sampling-based solvers. A general non-intrusive solver typically includes the following three steps:

- Step 1: one first generates N samples $\{\vec{\xi}^j\}_{j=1}^N$ for the random vector $\vec{\xi} \in \Omega$.
- Step 2: one solves the computational model $\mathbb{F}(\vec{x}(t,\vec{\xi}^j),\vec{\xi}^j) = 0$ at each sample $\vec{\xi}^j$. This step generates a set of realizations $\{\vec{x}(t,\vec{\xi}^j)\}_{j=1}^N$ and $\{y(\vec{\xi}^j)\}_{j=1}^N$.
- Step 3: one post-processes the obtained solution data $\{y(\vec{\xi}^j)\}_{j=1}^N$ to obtain a desired quantity of interest (e.g., the distribution, some statistical moments, or a close-form function approximation of $y(\vec{\xi})$).

The main advantage of a non-intrusive solver is its ease of implementation — a readily available deterministic solver (such as SPICE) can be used without any modifications to run the simulation for each sample. The samples for $\vec{\xi}$ can be generated in either a randomized or deterministic way, and different techniques can be employed in the post-processing steps. Based on the difference in the first and third steps, non-intrusive solvers can be further classified into different groups such as Monte Carlo [85–87], fitting/regression [88, 89], and stochastic collocation [54–57, 90].

Intrusive Solvers first convert the stochastic equation $\mathbb{F}(\vec{x}(t,\vec{\xi}),\vec{\xi}) = 0$ to a new deterministic problem (with a larger problem size) and then directly obtain $\vec{x}(t,\vec{\xi})$ by solving the new model **only once**. The main procedures are summarized below

- Step 1: one first approximates $\vec{x}(t,\vec{x})$ by the linear combination of K basis functions, i.e., $\vec{x}(t,\vec{\xi}) \approx \tilde{x}(t,\vec{\xi}) = \sum_{k=1}^{K} \hat{x}_k(t)\Psi_k(\vec{\xi})$.

- Step 2: one picks K testing functions $\{\Phi_k(\vec{\xi})\}_{k=1}^{K}$ that form a K-dimensional functional subspace. After that, a new deterministic computational model is built by enforcing $\mathbb{F}(\tilde{x}(t,\vec{\xi}),\vec{\xi})$ orthogonal to each testing function:

$$\left\langle \mathbb{F}(\tilde{x}(t,\vec{\xi}),\vec{\xi}), \Phi_k(\vec{\xi}) \right\rangle_{\Omega,\rho(\vec{\xi})} = 0 \text{ for } k = 1, \cdots, K. \tag{3.1}$$

  The resulting deterministic computational model can be written as

$$\mathcal{F}(\vec{\mathbf{x}}(t)) = 0 \in \mathbb{R}^{n\times K}, \tag{3.2}$$

  where $\vec{\mathbf{x}}(t) \in \mathbb{R}^{n\times K}$ includes the coefficients for all basis functions $\{\Psi_k(\vec{\xi})\}_{k=1}^{K}$.

- Step 3: one solves (3.2) to obtain $\vec{\mathbf{x}}(t)$ and the corresponding stochastic solution $\tilde{x}(t,\vec{\xi}) = \sum_{k=1}^{K} \hat{x}_k(t)\Psi_k(\vec{\xi})$.

The main advantage of an intrusive solver is that (3.2) needs to be simulated only once. On the other hand, simulating the resulting deterministic system may be expensive due to the possibly large problem size.

3.3 Monte Carlo Simulation

Monte Carlo [91] is a non-intrusive solver used in many fields, and it is implemented in almost all commercial circuit simulators. In Monte Carlo, N samples $\{\vec{\xi^j}\}_{j=1}^N$ are first generated according to PDF($\vec{\xi}$), the joint probability density function of $\vec{\xi}$. A deterministic solver is then called to simulate $\mathbb{F}(\vec{x}(t,\vec{\xi^j}),\xi^j) = 0$ at each sample, generating a set of deterministic solutions. Finally, all deterministic solutions are utilized to compute the statistical information of interest. For instance, for any quantity $y(\vec{\xi})$, its expectation value can be estimated as

$$\mathbb{E}(y(\vec{\xi})) \approx \frac{1}{N}\sum_{j=1}^{N} y(\vec{\xi^j}) \tag{3.3}$$

The error of Monte Carlo is asymptotically proportional to $\frac{1}{\sqrt{N}}$. Very often, a huge number (thousands to millions) of samples are required to achieve an acceptable level of accuracy. The excessive number of samples render the repeated simulation prohibitively expensive in many engineering applications.

3.4 Fitting and Regression Based Techniques

Fitting or regression techniques belong to the family of non-intrusive solvers. Given a set of samples $\{\vec{\xi^j}\}_{j=1}^N$ (normally picked in a randomized way) and the corresponding solution samples $\{y(\vec{\xi^j})\}_{j=1}^N$, an optimization or interpolation technique can be employed to determine an approximation $y(\vec{\xi}) \approx \tilde{y}(\vec{\xi}) = \sum_{k=1}^{K} \hat{y}_k \Psi_k(\vec{\xi})$. In other words, one can compute the coefficients of all basis functions by solving the linear equation

$$\mathbf{A}\hat{\mathbf{y}} = \mathbf{b} \tag{3.4}$$

where $\mathbf{A} \in \mathbb{R}^{N\times K}$ with $\mathbf{A}(j,k) = \Psi_k(\vec{\xi^j})$, $\hat{\mathbf{y}} \in \mathbb{R}^K$ with $\hat{\mathbf{y}}(k) = \hat{y}_k$, and $\mathbf{b} \in \mathbb{R}^N$ with $\mathbf{b}(j) = y(\vec{\xi^j})$. The post-processing techniques can be chosen dependent on the number of samples available:

- when $N>K$, (3.4) is over-determined, and thus one may solve a least-square problem to minimize $||\mathbf{A}\hat{\mathbf{y}} = \mathbf{b}||_2^2$, obtaining $\hat{\mathbf{y}} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$;
- when $N=K$ and $\mathbf{A}$ is invertible, one has $\hat{\mathbf{y}} = \mathbf{A}^{-1}\mathbf{b}$;
- when $N< K$, (3.4) is under-determined, and thus a regularization is required. In the machine learning community, regularized least-square is commonly utilized:

$$\hat{\mathbf{y}} = \text{argmin } ||\mathbf{A}\hat{\mathbf{y}} - \mathbf{b}||_2^2 + \lambda||\hat{\mathbf{y}}||_2^2. \tag{3.5}$$

In many engineering problems, $\hat{\mathbf{y}}$ has many zero elements, and thus one may use L_1 minimization to enforce the sparsity [92]

$$\hat{\mathbf{y}} = \text{argmin } ||\mathbf{A}\hat{\mathbf{y}} - \mathbf{b}||_2^2 + \lambda||\hat{\mathbf{y}}||_1, \tag{3.6}$$

where $||\mathbf{y}||_1 = \sum_{j=1}^{K} |\hat{\mathbf{y}}(j)|$. The regularization parameter $\lambda > 0$ is selected typically by cross validation.

3.5 Stochastic Collocation

Stochastic collocation [54–57] is the most popular non-intrusive stochastic spectral method. This approach also aims to approximate the state vector $\vec{x}(t, \vec{\xi})$ and/or output $y(\vec{\xi})$ by a linear combination of some basis functions. By using generalized polynomial chaos $H_k(\vec{\xi})$ as the basis function (i.e., letting $\Psi_k(\vec{\xi}) = H_k(\vec{\xi})$) and then exploiting the orthonormal property of $H_k(\vec{\xi})$, the weight of each basis function can be computed by a projection step. For instance

$$\hat{y}_k = \int_{\Omega} y(\vec{\xi})H_k(\vec{\xi})\text{PDF}(\vec{\xi})d\vec{\xi} \approx \sum_{j=1}^{\hat{N}} y(\vec{\xi}^j)H_k(\vec{\xi}^j)w^j, \tag{3.7}$$

where $\{\vec{\xi}^j, w^j\}_{j=1}^{\hat{N}}$ are picked by a proper quadrature rule. Similar to that in Monte Carlo and fitting/regression approaches, $y(\vec{\xi}^j)$ at each quadrature point is obtained by solving a deterministic problem $\mathbb{F}(\vec{x}(t, \vec{\xi}^j), \vec{\xi}^j) = 0$. In order to make stochastic

collocation efficient, one must use as few quadrature points as possible. For low-dimensional problems, the deterministic quadrature methods introduced in Chapter 2.2 are efficient. As the parameter dimensionality increases, such deterministic approaches quickly become intractable and thus one may have to utilize Monte-Carlo-type algorithms to estimate $\hat{y}_k$.

3.6 Stochastic Galerkin

Stochastic Galerkin is also called stochastic finite-element method [93] since it is a generalization of finite-element method to the multi-dimensional parameter space. It is an intrusive solver with special choice of basis functions and testing functions. First, the basis function $\Psi(\vec{\xi})$ is chosen as the generalized polynomial chaos, i.e., $\Psi_k(\vec{\xi}) = H_k(\vec{\xi})$. Second, a Galerkin projection is utilized to set up the deterministic model (3.2). Specifically, the testing functions are identical to the basis functions, i.e., $\Phi_k(\vec{\xi}) = \Psi_k(\vec{\xi}) = H_k(\vec{\xi})$.

Stochastic Galerkin is efficient for engineering problems. However, since the resulting equation is coupled, the efficiency of this technique can quickly degrade as the parameter dimensionality increases.

3.7 Previous Applications in Circuit and MEMS

Monte Carlo has been utilized in almost all commercial circuit simulators such as PSpice [5], Cadence Spectre [6], and Synopsys HSPICE [7]. Recent advancements include Mixture Importance Sampling, Quasi-Monte Carlo and Latin Hypercube sampling [85–87] that help reduce the number of simulation samples.

For low-dimensional problems, stochastic spectral methods are much more efficient over Monte Carlo, and thus they have become promising techniques in circuit/MEMS simulation. Ref. [12–17, 94–104] have applied Hermite polynomial-chaos expansion, stochastic collocation and stochastic Galerkin to simulate linear circuits with Gaussian variations. In order to handle high-dimensional linear problems, Moselhy has

developed a dominant singular-vector method [13] to accelerate intrusive solvers and a stochastic model reduction algorithm to accelerate non-intrusive solvers [105].

Only a small number of results have been reported for stochastic nonlinear circuits and MEMS problems. In [90, 106], stochastic spectral methods were applied to simulate nonlinear circuits with Gaussian variations. Manfredi [107] extended the idea of [106] to handle some problems with non-Gaussian variations, but new device models must be rebuilt according to the given uncertainty specifications and bias conditions. Pulch directly started from the equation (1.3) and applied stochastic Galerkin to simulate nonlinear RF circuits [108, 109] and multi-rate circuits [110]. The main limitation of a stochastic-Galerkin-based circuit simulator is its difficulty to solve the resulting coupled equations. In order to mitigate this issue, some techniques were developed to decouple the Jacobian matrices resulting from linear [111] and nonlinear circuit analysis [112] with Gaussian variations. The complexity of such techniques grows exponentially as the parameter dimensionality d increases (since $(p+1)^d$ basis functions were used where p is the highest polynomial degree for each random parameter). It is not clear if these techniques can handle non-Gaussian variations. In the MEMS community, there were a few work applying stochastic collocation to quantify the uncertainties caused by process variations [113–117].

Fitting and regression techniques have been applied for a long time for the stochastic behavior modeling of nonlinear circuits. The response-surface modeling in [1] uses monomials as the basis functions and then computes the coefficients of each monomial by a least-square optimization. Recently, Li [88] has applied compressed sensing [92] to obtain the sparse solutions for some high-dimensional problems.

Chapter 4

Stochastic Testing Simulator

This chapter presents a generalized polynomial chaos-based **intrusive** simulator, called **stochastic testing**, for the uncertainty quantification of transistor-level simulation. This SPICE-like stochastic simulator is a variant of the interpolation-based stochastic collocation [118, 119]. Our work uses a collocation testing method to set up a coupled equation such that the Jacobian matrix can be decoupled and thus the numerical computation can be accelerated. Our simulator differs from previous work in the following aspects:

1. Different from the non-intrusive stochastic collocation in [118, 119], this simulator is an intrusive framework: the resulting coupled equation is solved directly to obtain the spectral coefficients, **without** decoupling *a-priori*.

2. Different from stochastic collocation [54, 118] and stochastic Galerkin [93, 106] that use all quadrature points, our solver employs only a small portion of them to set up a deterministic equation. Our algorithm provides extra speedup in time-domain simulation, since its intrusive nature allows adaptive time stepping.

3. Decoupling is applied **inside** the Newton's iterations. Therefore, the computational complexity depends linearly on the total number of basis functions.

The above features make the proposed simulator hundreds to thousands of times faster over Monte Carlo, and tens to hundreds of times faster than stochastic Galerkin and stochastic collocation.

4.1 Stochastic Testing Simulator

4.1.1 Basic Idea of Stochastic Testing

First we approximate the exact solution $\vec{x}\left(t,\vec{\xi}\right)$ in stochastic differential algebraic equation (DAE) (1.3) by $\tilde{x}(t,\vec{\xi})$ with the truncated generalized polynomial chaos expansion in (2.8). This yields a residual function

$$\mathrm{Res}(\hat{\mathbf{x}}(t),\vec{\xi}) = \frac{d\vec{q}\left(\tilde{x}(t,\vec{\xi}),\vec{\xi}\right)}{dt} + \vec{f}\left(\hat{x}(t,\vec{\xi}),\vec{u}(t),\vec{\xi}\right). \tag{4.1}$$

Now the unknown vector reads

$$\hat{\mathbf{x}}\,(t) = [\hat{x}_1\,(t)\,;\,\cdots\,;\,\hat{x}_K\,(t)] \in \mathbb{R}^N, \text{ with } N = nK. \tag{4.2}$$

We set the highest total degree of the polynomial basis functions as p, and thus K is a polynomial function of p and d, and it is decided by (2.7).

In order to compute $\hat{\mathbf{x}}\,(t)$, stochastic testing starts from (4.1) and sets up a larger-size determined equation by the projection step (3.1). Recalling that stochastic Galerkin chooses testing functions as the basis functions and thus the resulting equation in (3.2) has larger-size coupled Jacobian matrices. In stochastic testing, we apply collocation testing by choosing the testing functions as

$$\Phi_k(\vec{\xi}) = \delta(\vec{\xi} - \vec{\xi^k}) \text{ for } k = 1,\cdots,K. \tag{4.3}$$

This choice of testing function avoids the multi-dimensional integral computation required in Galerkin testing. As a result, we obtain the following deterministic DAE

$$\frac{dQ\,(\hat{\mathbf{x}}\,(t))}{dt} + F\,(\hat{\mathbf{x}}\,(t),\vec{u}(t)) = 0 \tag{4.4}$$

with

$$Q(\hat{\mathbf{x}}(t)) = \begin{bmatrix} \vec{q}\left(\tilde{x}(t,\vec{\xi}^1),\vec{\xi}^1\right) \\ \vdots \\ \vec{q}\left(\tilde{x}(t,\vec{\xi}^K),\vec{\xi}^K\right) \end{bmatrix}, \; F(\hat{\mathbf{x}}(t),\vec{u}(t)) = \begin{bmatrix} \vec{f}\left(\tilde{x}(t,\vec{\xi}^1),\vec{u}(t),\vec{\xi}^1\right) \\ \vdots \\ \vec{f}\left(\tilde{x}(t,\vec{\xi}^K),\vec{u}(t),\vec{\xi}^K\right) \end{bmatrix}. \quad (4.5)$$

The collocation testing used here is the same with that used in collocation-based integral equation solvers [120]. However, in stochastic computation, "stochastic collocation" generally means a different sampling-based method (c.f. Section 4.2.2) that uses more sampling points than basis functions. Therefore, we name our proposed method as "stochastic testing".

There remain two important issues, and how to address them distinguishes our stochastic testing solver with the non-intrusive stochastic solvers in [118, 119]. The first issue is how to solve the resulting coupled DAE. Our solver directly solves (4.4) by an intrusive solver. As a result, the generalized polynomial chaos coefficients can be directly computed with adaptive time stepping [121]. The second issue is how to select the testing samples. Stochastic testing selects K testing points from some candidate nodes, whereas $(p+1)^d \gg K$ nodes are used in [118] to make the Jacobian invertible.

4.1.2 Intrusive Decoupled Solver

Instead of simulating each block of (4.4) separately, stochastic testing passes the whole coupled DAE into a specialized transient solver to directly compute the generalized polynomial chaos coefficients. Special matrix structures are exploited inside Newton's iterations to obtain simulation speedup. As a demonstration, we consider backward-Euler integration. Other types of numerical integration schemes (e.g., Trapezoidal or Gear-2 method) are implemented in a similar way inside stochastic testing simulator.

Let $\hat{\mathbf{x}}_k = \hat{\mathbf{x}}(t_k)$ and $\vec{u}_k = \vec{u}(t_k)$. In the transient solver, DAE (4.4) is discretized,

leading to an algebraic equation

$$\mathrm{R}(\hat{\mathbf{x}}_k) = \alpha_k(Q(\hat{\mathbf{x}}_k) - Q(\hat{\mathbf{x}}_{k-1})) + F(\hat{\mathbf{x}}_k, \vec{u}_k) = 0$$

with $\alpha_k = \frac{1}{t_k - t_{k-1}}$. The time step size is adaptively selected according to the local truncation error (LTE) [5, 121]. Starting from an initial guess $\hat{\mathbf{x}}_k^0$, $\hat{\mathbf{x}}_k$ is computed using Newton's iterations

$$\text{solve } \mathcal{J}\left(\hat{\mathbf{x}}_k^j\right) \Delta\hat{\mathbf{x}}_k^j = -\mathrm{R}\left(\hat{\mathbf{x}}_k^j\right), \text{ then update } \hat{\mathbf{x}}_k^{j+1} = \hat{\mathbf{x}}_k^j + \Delta\hat{\mathbf{x}}_k^j, \tag{4.6}$$

until convergence. Here $\mathcal{J}(\hat{\mathbf{x}}_k^j)$ is the Jacobian matrix of $\mathrm{R}(\hat{\mathbf{x}}_k^j)$. Fig. 4-1 shows the structure of $\mathcal{J}(\hat{\mathbf{x}}_k^j)$ from a CMOS low-noise amplifier (LNA) with n=14, d=p=3 and K=20. Clearly, all off-diagonal blocks are filled with non-zero submatrices. As a result, directly using a matrix solver to compute $\Delta\hat{\mathbf{x}}_k^j$ can be inefficient. If a direct matrix solver is employed, the linear system solution costs $O(N^3) = O(K^3n^3)$; when an iterative method is applied, the cost is $\hat{m}O(K^2n)$ where $\hat{m}$ is the number of iterations.

The coupled linear equation in (4.6) is instead solved in a decoupled manner. We rewrite the Jacobian matrix in (4.6) as

$$\mathcal{J}(\hat{\mathbf{x}}_k^j) = \tilde{\mathcal{J}}(\hat{\mathbf{x}}_k^j)\mathbf{W}_n, \text{ with } \mathbf{W}_n = \mathbf{V} \otimes \mathbf{I}_n \tag{4.7}$$

where $\mathbf{I}_n$ is the identity matrix of size $n \times n$, $\otimes$ denotes the Kronecker product operation, and $\mathbf{V} \in \mathbb{R}^{K\times K}$ is a Vandermonde-like matrix dependent only on the testing points and basis functions with the (j,k) element as

$$\mathbf{V}(j,k) = H_k(\vec{\xi}^j). \tag{4.8}$$

The inverse of $\mathbf{W}_n$ is

$$\mathbf{W}_n^{-1} = \mathbf{V}^{-1} \otimes \mathbf{I}_n \tag{4.9}$$

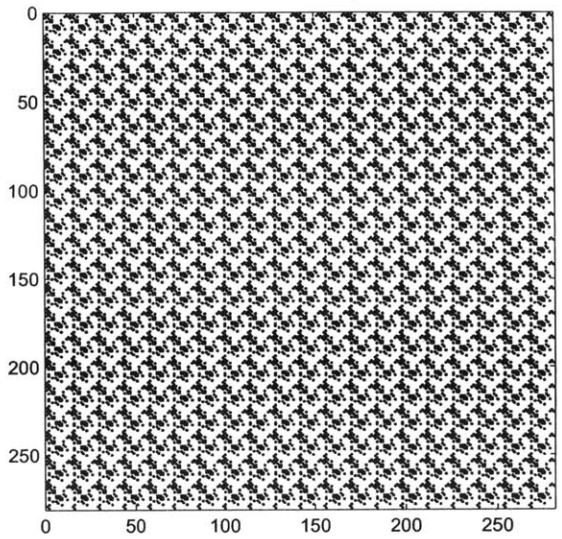


Figure 4-1: Structure of the Jacobian matrix in stochastic testing-based simulator, with $d = p = 3$ and $K = 20$.

which can be easily computed because: 1) $\mathbf{V}$ is of small size; and 2) fast inverse algorithms exist for Vandermonde-like matrices [122]. Both $\mathbf{V}$ and $\mathbf{V}^{-1}$ are calculated only once and then reused for all time points.

Matrix $\tilde{\mathcal{J}}(\hat{\mathbf{x}}_k^j)$ has a block-diagonal structure:

$$\tilde{\mathcal{J}}(\hat{\mathbf{x}}_k^j) = \begin{bmatrix} J(\hat{\mathbf{x}}_k^j, \vec{\xi}^1) & & \\ & \ddots & \\ & & J(\hat{\mathbf{x}}_k^j, \vec{\xi}^K) \end{bmatrix}. \tag{4.10}$$

Let $\hat{x}_{n_2}^{k,j}$ denotes the n_2-th generalized polynomial chaos coefficient vector in $\hat{\mathbf{x}}_k^j$, then

$$J(\hat{\mathbf{x}}_k^j, \vec{\xi}) = \alpha_k \frac{\partial \vec{q}(\vec{x}, \vec{\xi})}{\partial \vec{x}} + \frac{\partial \vec{f}(\vec{x}, \vec{u}_k, \vec{\xi})}{\partial \vec{x}}\bigg|_{\vec{x}=\sum_{n_2=1}^{K} \hat{x}_{n_2}^{k,j} H_{n_2}(\vec{\xi})}. \tag{4.11}$$

Finally, the linear equation in (4.6) is solved as follows:

1. Solve $\tilde{\mathcal{J}}(\hat{\mathbf{x}}_k^j)\Delta\mathbf{z} = -\mathrm{R}(\hat{\mathbf{x}}_k^j)$ for $\Delta\mathbf{z}$. Due to the block-diagonal structure, this step costs only $KO(n^3)$ for a direct solver or $\hat{m}KO(n)$ for an iterative solver.

2. Calculate the sparse matrix-vector product $\Delta\hat{\mathbf{x}}_k^j = \mathbf{W}_n^{-1}\Delta\mathbf{z}$. Since the closed form of $\mathbf{W}_n^{-1}$ is ready, the matrix-vector multiplication costs only $O(nK)$.

The computational cost of stochastic testing solver now has only a linear dependence on K, as contrasted with the cubic or quadratic dependence when directly solving the coupled linear equation.

The stochastic testing solver can be easily implemented inside a commercial circuit simulator without any modifications to device models. Inside each Newton's iteration, one can convert $\hat{\mathbf{x}}_k^j$ to a deterministic state vector and then evaluate the corresponding Jacobian and function values for a testing sampling. Repeating this procedure for all samples, $\tilde{\mathcal{J}}(\hat{\mathbf{x}}_k^j)$ and $\mathrm{R}(\hat{\mathbf{x}}_k^j)$ can be obtained. After that, all blocks are solved independently to obtain $\Delta\mathbf{z}$ and then $\Delta\hat{\mathbf{x}}_k^j$. If the Newton's iterations get converged, the local truncation error (LTE) is checked by an existing estimator [5, 121]. The solution is accepted and stochastic testing proceeds to the next time point if the LTE is below a threshold; otherwise, the time step size is reduced and $\hat{\mathbf{x}}_k$ is recomputed. Since the function/Jacobian evaluation and linear system solutions are well decoupled, stochastic testing can be easily implemented on a parallel computing platform.

4.1.3 Testing Sample Selection

The testing samples $\{\vec{\xi}^j\}_{j=1}^K$ are selected by two steps. First, $(p+1)^d$ candidate samples are generated by a Gaussian-quadrature tensor product rule. Next, only K samples (with $K \ll p+1)^d$) are selected from the candidate samples and used as the final testing samples. Note that $(p+1)^d$ samples are used in [118], which are exactly the candidate samples of stochastic testing.

Candidate Sample Generation

In this work, we set $\hat{n} = p+1$ (p is highest total polynomial order), and use Gauss quadrature rule described in Chapter 2.2.1 to construct quadrature points for each random parameter ξ_k. Then, we apply a tensor product or sparse grid rule to generate multirate quadrature points as the candidate samples. For instance, we apply the

tensor-product rule in (2.14) to construct $\hat{N} = \hat{n}^d$ quadrature samples in the d-dimensional parameter space Ω. Define an index matrix $\mathcal{I} \in \mathbb{Z}^{d \times \hat{N}}$, the j-th column of which is decided according to the constraint

$$1 + \sum_{k=1}^{d} (\hat{n} - 1)^{k-1} (\mathcal{I}(k, j) - 1) = j. \tag{4.12}$$

Then the j-th quadrature point in Ω is

$$\vec{\xi}_j = [\xi_1^{\mathcal{I}(1,j)}, \cdots, \xi_l^{\mathcal{I}(d,j)}], \tag{4.13}$$

where $1 \leq \mathcal{I}(k, j) \leq \hat{n}$ indicates the index of the quadrature point in Ω_k. The corresponding weight of $\vec{\xi}_j$ can also be rewritten as

$$w^j = \prod_{k=1}^{d} w_k^{\mathcal{I}(k,j)}. \tag{4.14}$$

Selecting Testing Samples

In the second step, only K testing samples are selected from the $(p+1)^d$ candidate samples based on two criteria:

1. We prefer those quadrature points that are statistically "important", i.e., those samples with large weight values;

2. The matrix $\mathbf{V}$ defined in (4.8) should be full-rank and well conditioned.

The Matlab pseudo codes of selecting the final testing samples are provided in Alg. 1. In Line 7, $\beta > 0$ is a threshold scalar. The input vector in Line 2 is $\vec{w}$=$[|w^1|, |w^2|, \cdots, |w^{\hat{N}}|]$, and the vector-valued function $\vec{H}(\xi) \in \mathbb{R}^{K \times 1}$ is

$$\vec{H}(\vec{\xi}) = [H_1(\vec{\xi}), H_2(\vec{\xi}), \cdots, H_K(\vec{\xi})]^T. \tag{4.15}$$

The basic idea of Alg. 1 is as follows. All candidate samples and their weights are reordered such that $|w^j| \geq |w^{j+1}|$, and the first sample is selected as the first

Algorithm 1 Testing Node Selection.

Construct $\hat{N}$ d-dimensional Gaussian quadrature samples and weights;
$[\vec{w}, \text{ind}]$=sort($\vec{w}$, 'descend'); % reorder the weights
$V = \vec{H}\left(\vec{\xi}_k\right) / ||\vec{H}\left(\vec{\xi}_k\right)||$, with $k = \text{ind}(1)$;
$\vec{\xi}^1 = \vec{\xi}_k$, $m = 1$; % the 1st testing sample
for $j = 2, \cdots, \hat{N}$ **do**
  $k = \text{ind}(j)$, $\vec{v} = \vec{H}\left(\vec{\xi}_k\right) - \mathbf{V}_m\left(\mathbf{V}_m^T \vec{H}\left(\vec{\xi}_k\right)\right)$;
  if $||\vec{v}||/||\vec{H}\left(\vec{\xi}_k\right)|| > \beta$
    $\mathbf{V}_{m+1} = [\mathbf{V}_m; \vec{v}/||\vec{v}||]$, $m = m + 1$;
    $\vec{\xi}^m = \vec{\xi}_k$; % select as a new testing sample.
    if $m \geq K$, break, **end**;
  end if
end for

testing sample $\vec{\xi}^1$. Then, we consider the remaining candidate samples from the "most important" to the "least important". Assuming that $m - 1$ testing samples have been selected, this defines a vector space

$$\mathbf{V}_{m-1} = \text{span}\left\{\vec{H}(\vec{\xi}^1), \cdots, \vec{H}(\vec{\xi}^{m-1})\right\}. \tag{4.16}$$

The next "most important" candidate $\vec{\xi}_k$ is selected as a new testing sample if and only if $\vec{H}(\vec{\xi}_k)$ has a large enough component orthogonal to $\mathbf{V}_{m-1}$. This means that we can add one dimension to $\mathbf{V}_{m-1}$ by choosing $\vec{\xi}_k$ as a new testing point, leading to a new vector space $\mathbf{V}_m$. This procedure continues until the dimensionality of $\mathbf{V}_m$ becomes K.

When the parameter dimensionality d is large, generating and saving the candidate samples and index matrix $\mathcal{I}$ become expensive. A solution is to select the testing samples without explicitly generating the candidate samples or $\mathcal{I}$. First, we generate weight w^j's and the corresponding index j's according to (4.14) and (4.12), respectively. In the k-th step, we find the k-th largest weight w^j and its corresponding index j. According to (4.12), the j-th column of the index matrix $\mathcal{I}$ can be calculated, and then we can construct candidate sample $\vec{\xi}_j$. Finally $\vec{\xi}_j$ is selected as a new testing sample $\vec{\xi}^m$ if $\vec{H}(\vec{\xi}_j)$ has a large enough component orthogonal to $\mathbf{V}_{m-1}$, otherwise it

is omitted and not stored.

There exist other possible ways to select the testing samples. A recent progress is to generate the samples by Leja sequences, a greedy approximation to Fekete nodes [123]. How to select the optimal testing samples is still an open problem.

4.2 Comparison with Other Stochastic Solvers

Now we briefly extends the generalized polynomial chaos-based stochastic Galerkin and stochastic collocation to nonlinear circuit problems, and then we compare them with our proposed stochastic testing simulator.

4.2.1 Comparison with Stochastic Galerkin

Stochastic Galerkin for Nonlinear Circuits. Starting from the residual function (4.1), stochastic Galerkin sets up a deterministic DAE in the form (4.4) by Galerkin testing:

$$\left\langle \mathrm{Res}\left(\hat{\mathbf{x}}(t),\vec{\xi}\right), H_k\left(\vec{\xi}\right)\right\rangle_{\Omega,\rho(\vec{\xi})} = 0, \text{ for } k = 1,\cdots,K. \tag{4.17}$$

Now $Q(\hat{\mathbf{x}}(t))$ and $F(\hat{\mathbf{x}}(t),\vec{u}(t))$ in (4.4) have the blocked form

$$Q\left(\hat{\mathbf{x}}(t)\right) = \begin{bmatrix} Q_1\left(\hat{\mathbf{x}}(t)\right) \\ \vdots \\ Q_K\left(\hat{\mathbf{x}}(t)\right) \end{bmatrix}, \; F\left(\hat{\mathbf{x}}(t),\vec{u}(t)\right) = \begin{bmatrix} F_1\left(\hat{\mathbf{x}}(t),\vec{u}(t)\right) \\ \vdots \\ F_K\left(\hat{\mathbf{x}}(t),\vec{u}(t)\right) \end{bmatrix}, \tag{4.18}$$

with the n_1-th block defined by

$$\begin{aligned} Q_{n_1}\left(\hat{\mathbf{x}}\left(t\right)\right) &= \left\langle \vec{q}\left(\hat{x}(t,\vec{\xi}),\vec{\xi}\right), H_{n_1}(\vec{\xi})\right\rangle_{\Omega,\rho(\vec{\xi})}, \\ F_{n_1}\left(\hat{\mathbf{x}}\left(t\right),\vec{u}(t)\right) &= \left\langle \vec{f}\left(\hat{x}(t,\vec{\xi}),\vec{u}(t),\vec{\xi}\right), H_{n_1}(\vec{\xi})\right\rangle_{\Omega,\rho(\vec{\xi})}. \end{aligned} \tag{4.19}$$

In order to obtain the above inner product, one can use deterministic numerical quadrature or Monte Carlo integration [124].

Comparison with Our Simulator. Both techniques are intrusive solvers, and

the coupled DAEs from stochastic testing and stochastic Galerkin have the same size. However, stochastic Galerkin is more expensive compared to stochastic testing. First, stochastic Galerkin must evaluate the multivariate stochastic integrals in (4.19), hence functions $\vec{q}$ and $\vec{f}$ must be evaluated at many quadrature or sampling points. This step is not cheap because evaluating a semiconductor device model (e.g., BISM3 model) at each sample involves running tens of thousands of lines of codes. Second, the linear system solving inside the Newton's iteration of stochastic Galerkin is much more expensive. Assume that Gaussian quadrature is applied to calculate the inner products in (4.19), then the Jacobian $\mathcal{J}(\hat{\mathbf{x}}_k^j)$ has the following structure

$$\mathcal{J}\left(\hat{\mathbf{x}}_k^j\right) = \begin{bmatrix} \mathcal{J}_{1,1}\left(\hat{\mathbf{x}}_k^j\right) & \cdots & \mathcal{J}_{1,K}\left(\hat{\mathbf{x}}_k^j\right) \\ \vdots & \ddots & \vdots \\ \mathcal{J}_{K,1}\left(\hat{\mathbf{x}}_k^j\right) & \cdots & \mathcal{J}_{K,K}\left(\hat{\mathbf{x}}_k^j\right) \end{bmatrix}, \tag{4.20}$$

and the submatrix $\mathcal{J}_{n_1,n_2}\left(\hat{\mathbf{x}}_k^j\right) \in \mathbb{R}^{n\times n}$ is calculated by

$$\mathcal{J}_{n_1,n_2}\left(\hat{\mathbf{x}}_k^j\right) = \sum_{q=1}^{\hat{N}} w^q H_{n_1}\left(\vec{\xi}^q\right) H_{n_2}\left(\vec{\xi}^q\right) J\left(\hat{\mathbf{x}}_k^j, \vec{\xi}^q\right).$$

Here $\vec{\xi}^q$ is the q-th Gaussian quadrature sample and w^q the corresponding weight, $J\left(\hat{\mathbf{x}}_k^j, \vec{\xi}^q\right)$ is calculated according to the definition in (4.11). The Jacobian in stochastic Galerkin cannot be decoupled. Therefore, solving the resulting DAE of stochastic Galerkin requires $O(N^3) = O(K^3n^3)$ at each time point if a direct solver is used (or $\hat{m}O(K^2n)$ if $\hat{m}$ iterations are used in an iterative solver), much more expensive compared to stochastic testing.

4.2.2 Comparison with Stochastic Collocation Method

Stochastic Collocation for Nonlinear Circuits. Obeying the procedures in Chapter 3.5, stochastic collocation starts from the original stochastic circuit equation (1.3) without using generalized polynomial chaos approximation *a-priori*. With $\hat{N}$ quadrature points $\vec{\xi}^1, \cdots, \vec{\xi}^{\hat{N}}$, stochastic collocation solves (1.3) at each sample to

obtain a deterministic solution $\vec{x}(t, \vec{\xi}^k)$. The generalized polynomial chaos coefficients are then computed using a post-processing such as projection

$$\hat{x}_j(t) = \left\langle \vec{x}(t, \vec{\xi}), H_j(\vec{\xi}) \right\rangle_{\Omega, \rho(\vec{\xi})} \approx \sum_{k=1}^{\hat{N}} w^k H_j(\vec{\xi}^k) \vec{x}(t, \vec{\xi}^k). \tag{4.21}$$

Comparison with Our Simulator. Similar to stochastic testing, the cost of stochastic collocation has a linear dependence on the number of samples used. However, stochastic collocation uses more samples than stochastic testing. Furthermore, stochastic collocation is not as efficient as stochastic testing in time-domain simulation. In order to reconstruct the generalized polynomial chaos coefficients of the time-domain state vector $\vec{x}(t, \vec{\xi})$, stochastic collocation must use the same time grid points to simulate all deterministic circuit equations. Since it is difficult to preselect an adaptive time grid, a small fixed step size is normally used, leading to excessive computational cost. In contrast, stochastic testing can use any standard adaptive step stepping to accelerate the time-domain simulation since it directly computes the generalized polynomial chaos coefficients. It seems that stochastic collocation can use adaptive time stepping to simulate each deterministic equation, and then uses interpolation at the time points where solutions are missing. Unfortunately, the errors caused by such interpolations are much larger than the threshold inside Newton's iterations, causing inaccurate computation of higher-order generalized polynomial chaos coefficients. However, stochastic collocation indeed can use adaptive time stepping if one is not interested in the statistical information of the time-domain waveforms.

4.2.3 Summary and Comparison

All stochastic spectral methods are summarized in Table 4.1. Stochastic testing allows both adaptive time stepping and decoupled simulation, therefore, it is more efficient over stochastic collocation and stochastic Galerkin for circuit simulation.

Table 4.1: Comparison of different spectral methods.

Method	Type	Decoupled?	Adapt. step size?
stochastic collocation	nonintrusive	√	×
stochastic Galerkin	intrusive	×	√
stochastic testing	intrusive	√	√

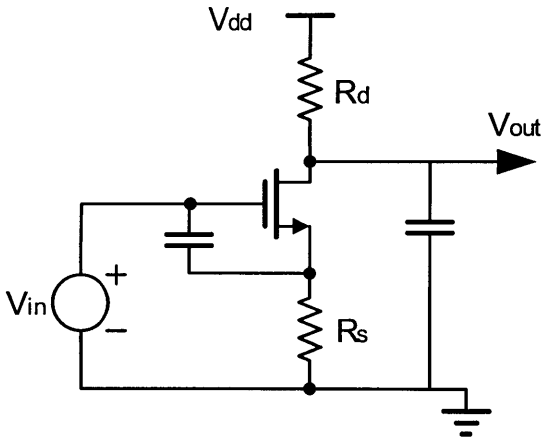


Figure 4-2: Schematic of the common-source amplifier.

4.3 Numerical Results

This section presents the simulation results of some analog, RF and digital integrated circuits. Our stochastic testing algorithm is implemented in a MATLAB prototype simulator and integrated with several semiconductor device models for algorithm verification. In this work, Level-3 MOSFET model and Ebers-Moll BJT model are used for transistor evaluation [125]. The TSMC 0.25μm CMOS model card [126] is used to describe the device parameters of all MOSFETs. For simplicity, in this section, **we use ST, SC, SG and MC to represent stochastic testing, stochastic collocation, stochastic Galerkin and Monte Carlo, respectively**. In SG and ST, step sizes are selected adaptively according to the local truncation error (LTE) [121] for time-domain simulation. In contrast, uniform step sizes are used for both MC and SC since we need to obtain the statistical information of time-domain solutions. In our experiments, all candidate samples of stochastic testing are generated by Gaussian quadrature and tensor-product rules. The cost of generating the candidate samples

and selecting testing samples is several milliseconds, which is negligible. For all circuit examples, SC and SG use the samples from a tensor-product rule. The sparse-grid and tensor-product SC methods are compared with ST in detail in Section 4.3.6.

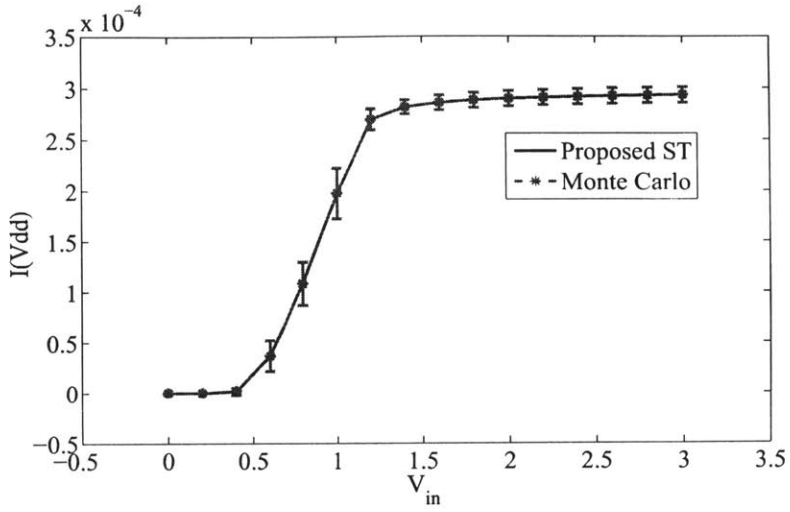


Figure 4-3: Error bars showing the mean and s.t.d values from our stochastic testing method (blue) and Monte Carlo method (red) of $I(V_{dd})$.

4.3.1 Illustrative Example: Common-Source (CS) Amplifier

The common-source (CS) amplifier in Fig. 4-2 is used to compare comprehensively our stochastic testing-based simulator with MC and other stochastic spectral methods. This amplifier has 4 random parameters: 1) V_T (threshold voltage when $V_{bs} = 0$) has a normal distribution; 2) temperate T has a shifted and scaled Beta distribution, which influences V_{th}; 3) R_s and R_d have Gamma and uniform distributions, respectively.

Stochastic Testing versus Monte Carlo

Stochastic testing method is first compared with MC in DC sweep. By sweeping the input voltage from 0 V up to 3 V with a step size of 0.2 V, we estimate the supply currents and DC power dissipation. In MC, 10^5 sampling points are used. In our

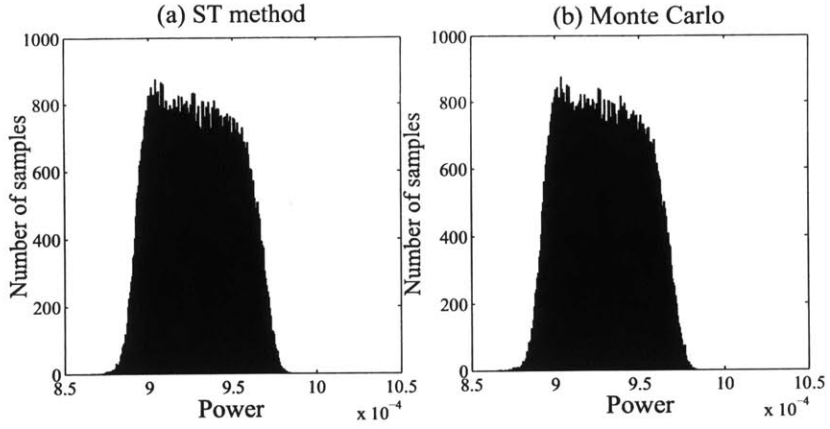


Figure 4-4: Histograms showing the distributions of the power dissipation at $V_{\text{in}} = 1.4\text{V}$, obtained by stochastic testing method (left) and Monte Carlo (right).

stochastic testing simulator, using an order-3 truncated generalized polynomial chaos expansion (with 35 generalized polynomial chaos basis functions, and 35 testing samples selected from 256 candidate samples) achieves the same level of accuracy. The error bars in Fig. 4-3 show that the mean and s.t.d values from both methods are indistinguishable. The histograms in Fig. 4-4 plots the distributions of the power dissipation at $V_{\text{in}} = 1.4\text{V}$. Again, the results obtained by stochastic testing is consistent with MC. The expected value at 1.4V is 0.928 mW from both methods, and the s.t.d. value is 22.07 μW from both approaches. Apparently, the variation of power dissipation is not a Gaussian distribution due to the presence of circuit nonlinearity and non-Gaussian random parameters.

CPU times: For this DC sweep, MC costs about 2.6 hours, whereas our stochastic testing simulator only costs 5.4 seconds. Therefore, a 1700$\times$ speedup is achieved by using our stochastic testing simulator.

Stochastic Testing versus SC and SG in DC Analysis

Next, stochastic testing method is compared with SG and SC. Specifically, we set $V_{\text{in}} = 1.6\text{V}$ and compute the generalized polynomial chaos coefficients of all state variables with the total generalized polynomial chaos order p increasing from 1 to 6. We use the results from $p = 6$ as the "exact solution" and plot the L_2 norm of the

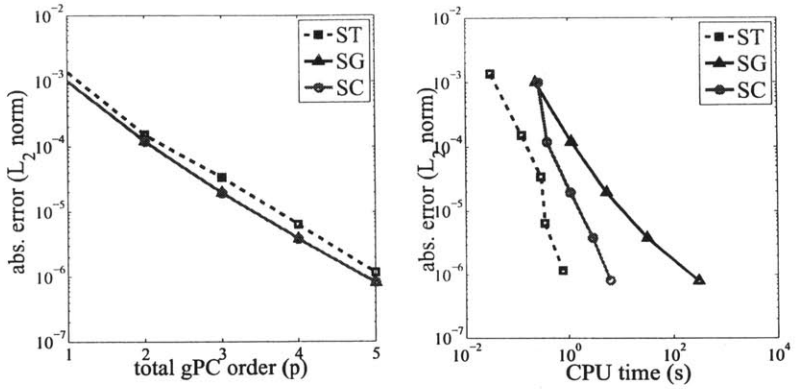


Figure 4-5: Absolute errors (measured by L_2 norm) of the generalized polynomial chaos coefficients for the DC analysis of the CS amplifier, with $V_{\text{in}} = 1.6$V. Left: absolute errors versus generalized polynomial chaos order p. Right: absolute errors versus CPU times.

Table 4.2: Computational cost of the DC analysis for CS amplifier.

gPC order (p)		1	2	3	4	5	6
ST	time (s)	0.16	0.22	0.29	0.51	0.78	1.37
	# samples	5	15	35	70	126	210
SC	time (s)	0.23	0.33	1.09	2.89	6.18	11.742
	# samples	16	81	256	625	1296	2401
SG	time (s)	0.25	0.38	5.33	31.7	304	1283
	# samples	16	81	256	625	1296	2401

absolute errors of the computed generalized polynomial chaos coefficients versus p and CPU times, respectively. The left part of Fig. 4-5 shows that as p increases, ST, SC and SG all converge very fast. Although ST has a slightly lower convergence rate, its error still rapidly reduces to below 10^{-4} when $p = 3$. The right part of Fig. 4-5 shows that ST costs the least CPU time to achieve the same level of accuracy with SC and SG, due to the decoupled Newton's iterations and fewer samples used in ST.

CPU times: The computational costs of different solvers are summarized in Table 4.2. The speedup of ST becomes more significant as the total generalized polynomial chaos order p increases. We remark that the speedup factor will be smaller if SC uses sparse grids, as will be discussed in Section 4.3.6.

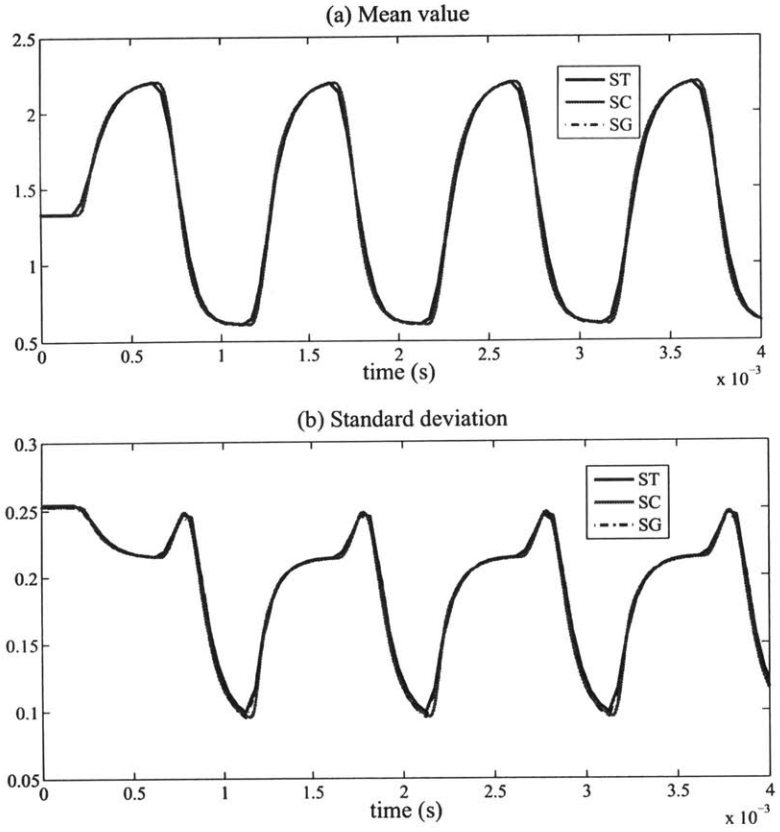


Figure 4-6: Transient waveform of the output of the CS amplifier.

Stochastic Testing versus SC and SG in Transient Simulation

Table 4.3: Computational cost of transient simulation for CS amplifier.

Methods	ST	SG	SC
CPU times	41 s	> 1 h	1180 s
# samples	35	256	256
speedup of ST	1	> 88	29

Finally, ST is compared with SG and SC in transient simulation. It is well known that the SG method provides an optimal solution in terms of accuracy [4, 40, 41], therefore, the solution from SG is used as the reference for accuracy comparison. The total generalized polynomial chaos order is set as $p = 3$ (with $K = 35$ testing samples selected from 256 candidate samples), and the Gear-2 integration scheme [121] is

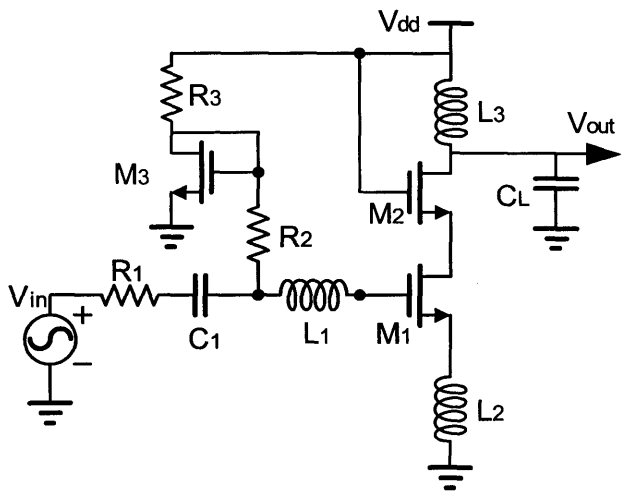


Figure 4-7: Schematic of the LNA.

used for all spectral methods. In SC, a uniform step size of 10μs is used, which is the largest step size that does not cause simulation failures. The input is kept as $V_{\text{in}} = 1$ V for 0.2 ms and then added with a small-signal square wave (with 0.2V amplitude and 1 kHz frequency) as the AC component. The transient waveforms of V_{out} are plotted in Fig. 4-6. The mean value and standard deviation from ST are almost indistinguishable with those from SG.

It is interesting that the result from ST is more accurate than that from SC in this transient simulation example. This is because of the employment of LTE-based step size control [121]. With a LTE-based time stepping [121], the truncation errors caused by numerical integration can be well controlled in ST and SG. In contrast, SC cannot adaptively select the time step sizes according to LTEs, leading to larger integration errors.

CPU times: The computational costs of different solvers are summarized in Table 4.3. It is noted that SC uses about 7× of samples of ST, but the speedup factor of ST is 29. This is because the adaptive time stepping in ST causes an extra speedup factor of about 4. MC is prohibitively expensive for transient simulation and thus not compared here.

Table 4.4: Computational cost of the DC analysis for LNA.

gPC order (p)		1	2	3	4	5	6
ST	time (s)	0.24	0.33	0.42	0.90	1.34	2.01
	# samples	4	10	20	35	56	84
SC	time (s)	0.26	0.59	1.20	2.28	4.10	6.30
	# samples	8	27	64	125	216	343
SG	time (s)	0.58	2.00	6.46	24.9	87.2	286
	# samples	8	27	64	125	216	343

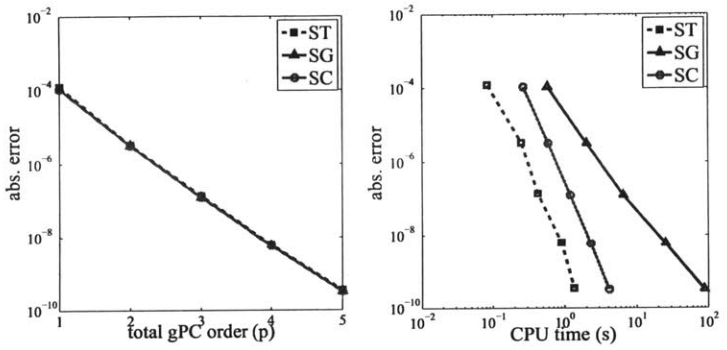


Figure 4-8: Absolute errors (measured by L_2 norm) of the generalized polynomial chaos coefficients for the DC analysis of LNA. Left: absolute errors versus generalized polynomial chaos order p. Right: absolute errors versus CPU times.

4.3.2 Low-Noise Amplifier (LNA)

Now we consider a practical low-noise amplifier (LNA) shown in Fig 4-7. This LNA has 3 random parameters in total: resistor R_3 is a Gamma-type variable; R_2 has a uniform distribution; the gate width of M_1 has a uniform distribution.

DC Analysis: We first run DC analysis by ST, SC and SG with p increasing from 1 to 6, and plot the errors of the generalized polynomial chaos coefficients of the state vector versus p and CPU times in Fig. 4-8. For this LNA, ST has almost the same accuracy with SC and SG, and it requires the smallest amount of CPU time. The cost of the DC analysis is summarized in Table 4.4.

Transient Analysis: An input signal $V_{\text{in}} = 0.5\sin(2\pi f t)$ with $f = 10^8$ Hz is added to this LNA. We are interested in the uncertainties of the transient waveform at the output. Setting $p = 3$, our ST method uses 20 generalized polynomial chaos basis functions (with 20 testing samples selected from 64 candidate samples) to obtain

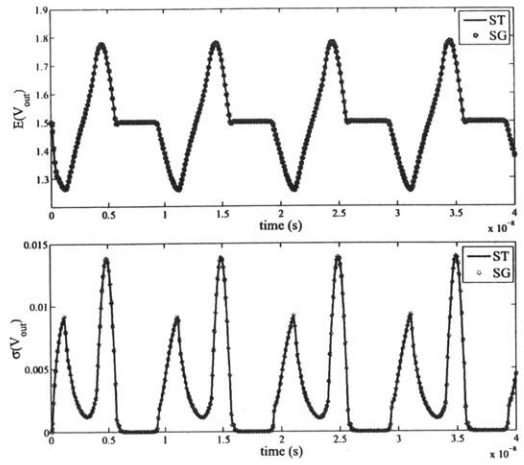


Figure 4-9: Transient simulation results of the LNA. Upper part: expectation of the output voltage; bottom part: standard deviation of the output voltage.

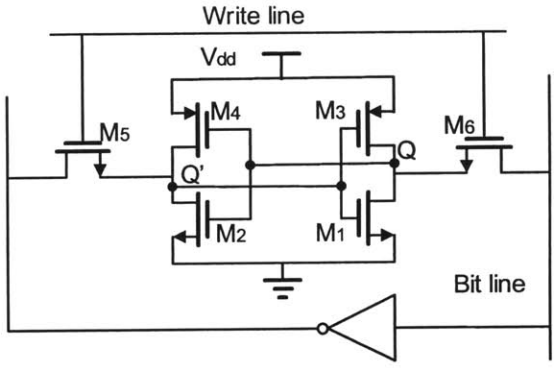


Figure 4-10: Schematic of the CMOS 6-T SRAM.

the waveforms of the first 4 cycles. The result from ST is indistinguishable with that from SG, as shown in Fig. 4-9. ST consumes only 56 seconds for this LNA. Meanwhile, SG costs 26 minutes, which is 28× slower compared to ST.

4.3.3 6-T SRAM Cell

The 6-T SRAM cell in Fig. 4-10 is studied to show the application of ST in digital cell analysis. When the write line has a high voltage (logic 1), the information of the bit line can be written into the cell and stored on transistors $M_1 - M_4$. The 1-bit

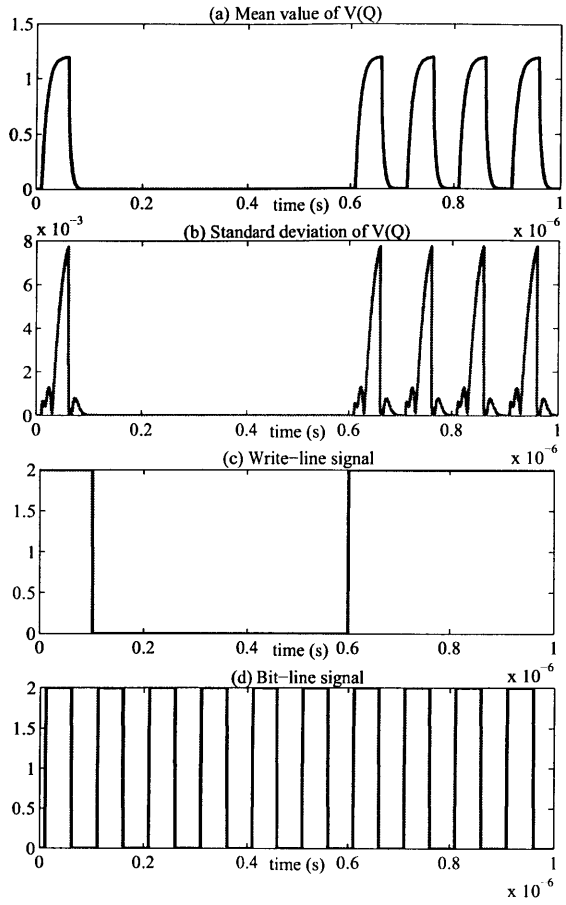


Figure 4-11: Uncertainties of the SRAM cell. (a) and (b) shows the expectation and standard deviation of V_{out}; (c) and (d) shows the waveforms of the write line and bit line, respectively.

information is represented by the voltage of node Q. When the write line has a low voltage (logic 0), M_5 and M_6 turn off. In this case, $M_1 - M_4$ are disconnected with the bit line, and they form a latch to store and hold the state of node Q. Here V_{dd} is set as 1 V, while the high voltages of the write and bit lines are both set as 2 V.

Now we assume that due to mismatch, the gate widths of $M_1 - M_4$ have some variations which can be expressed as Gaussian variables. Here we study the influence of device variations on the transient waveforms, which can be further used for power and timing analysis. Note that in this paper we do not consider the rare failure events of SRAM cells [85]. In order to quantify the uncertainties of the voltage waveform

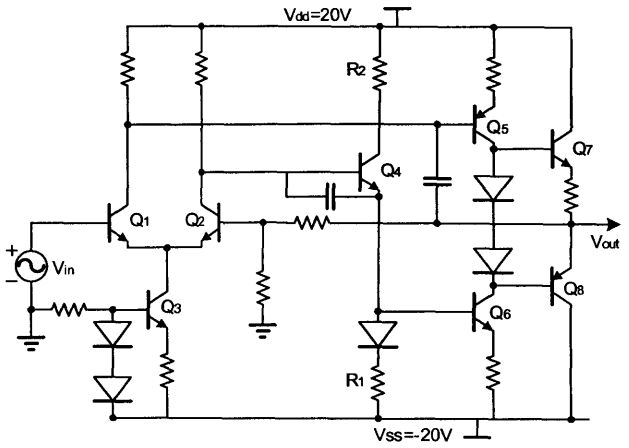


Figure 4-12: Schematic of the BJT feedback amplifier.

at sample Q, our ST method with $p = 3$ and $K = 35$ (with 35 testing samples selected from 256 candidate samples) is applied to perform transient simulation under a given input waveforms. Fig. 4-11 shows the waveforms of write and bit lines and the corresponding uncertainties during the time interval $[0, 1]\mu$s.

CPU times: Our ST method costs 6 minutes to obtain the result. SG generates the same results at the cost of several hours. Simulating this circuit with SC or MC is prohibitively expensive, as a very small uniform step size must be used due to the presence of sharp state transitions.

4.3.4 BJT Feedback Amplifier

In order to show the application of our ST method in AC analysis and in BJT-type circuits, we consider the feedback amplifier in Fig. 4-12. In this circuit, R_1 and R_2 have Gamma-type uncertainties. The temperature is a Gaussian variable which significantly influences the performances of BJTs and diodes. Therefore, the transfer function from V_{in} to V_{out} is uncertain.

Using $p = 3$ and $K = 20$ (with 20 testing samples selected from 64 candidate samples), our ST simulator achieves the similar level of accuracy of a MC simulation using 10^5 samples. The error bars in Fig. 4-13 show that the results from both methods are indistinguishable. In ST, the real and imaginary parts of the transfer

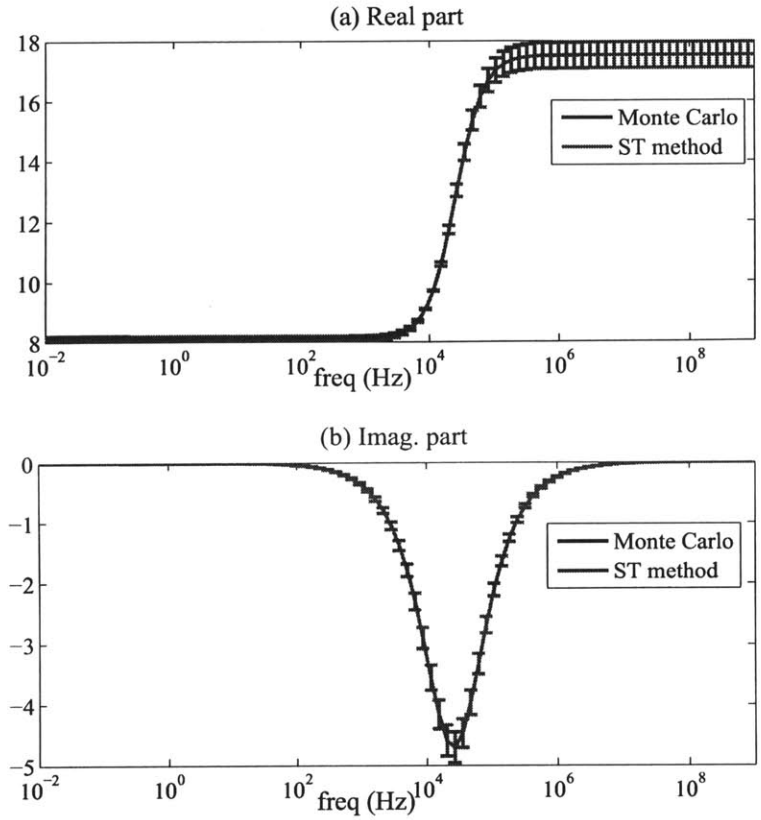


Figure 4-13: Uncertainties of the transfer function of the BJT amplifier.

functions are both obtained as truncated generalized polynomial chaos expansions. Therefore, the signal gain at each frequency point can be easily calculated with a simple polynomial evaluation. Fig. 4-14 shows the calculated PDF of the small-signal gain at $f = 8697.49$ Hz using both ST and MC. The PDF curves from both methods are indistinguishable.

CPU times: The simulation time of ST and Monte Carlo are 3.6 seconds and over 2000 seconds, respectively.

4.3.5 BJT Double-Balanced Mixer

As the final circuit example, we consider the time-domain simulation of RF circuits excited by multi-rate signals, by studying the double-balanced mixer in Fig. 4-15.

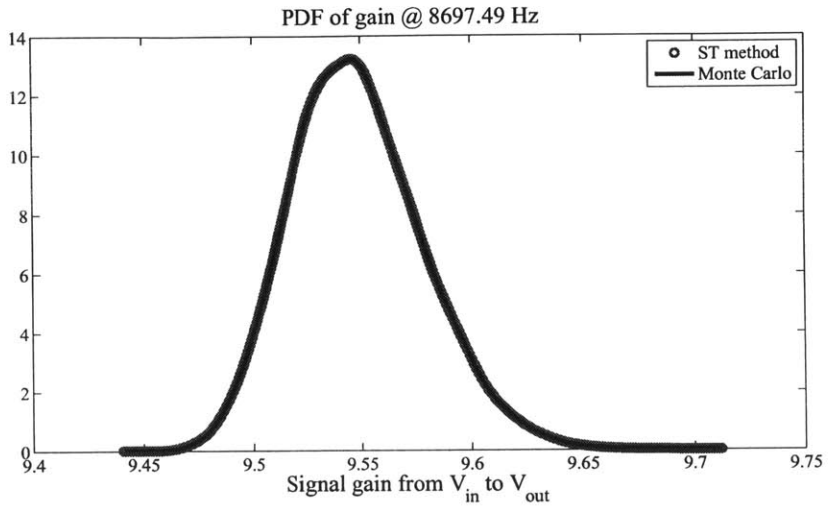


Figure 4-14: Simulated probability density functions of the signal gain.

Transistors Q_1 and Q_2 accept an input voltage of frequency f_1, and $Q_3 \sim Q_6$ accept the second input of frequency f_2. The output $v_{\text{out}} = V_{\text{out1}} - V_{\text{out2}}$ will have components at two frequencies: one at $|f_1 - f_2|$ and the other at $f_1 + f_2$. Now we assume that R_1 and R_2 are both Gaussian-type random variables, and we measure the uncertainties of the output voltage. In our simulation, we set $V_{\text{in1}} = 0.01\sin(2\pi f_1 t)$ with $f_1 = 4$ MHz and $V_{\text{in2}} = 0.01\sin(2\pi f_2 t)$ with $f_2 = 100$ kHz. We set $p = 3$ and $K = 10$ (with 10 testing samples selected from 16 candidate samples), and then use our ST simulator to run a transient simulation from $t = 0$ to $t = 30\mu$s. The expectation and standard deviation of $V_{\text{out1}} - V_{\text{out2}}$ are plotted in Fig. 4-16.

CPU times: The cost of our ST method is 21 minutes, whereas simulating this mixer by SG, SC or MC on the same MATLAB platform is prohibitively expensive due to the presence of multi-rate signals and the large problem size.

4.3.6 Discussion: Speedup Factor of ST over SC

Finally we comprehensively compare the costs of ST and SC. Two kinds of SC methods are considered according to the sampling samples used in the solvers [43]: SC using tensor product (denoted as SC-TP) and SC using sparse grids (denoted as SC-SP). SC-TP uses $(p+1)^d$ samples to reconstruct the generalized polynomial chaos

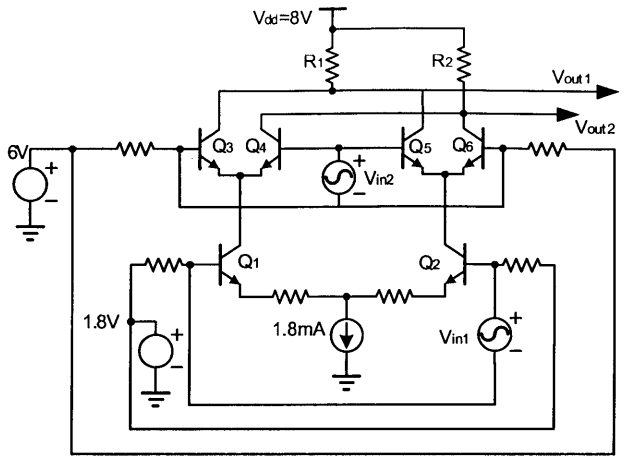


Figure 4-15: Schematic of the BJT double-balanced mixer.

coefficients, and the work in [118] belongs to this class. For SC-SP, a level-$p+1$ sparse grid must be used to obtain the p-th-order generalized polynomial chaos coefficients in (4.21). We use the Fejèr nested sparse grid in [42], and according to [127] the total number of samples in SC-SP is estimated as

$$N_{\text{SC-SP}} = \sum_{i=0}^{p} 2^i \frac{(d-1+i)!}{(d-1)!i!} \tag{4.22}$$

DC Analysis: In DC analysis, since both ST and SC use decoupled solvers and their costs linearly depend on the number of samples, the speedup factor of ST versus SC is

$$\nu_{\text{DC}} \approx N_{\text{SC}}/K \tag{4.23}$$

where N_{SC} and K are the the numbers of samples used by SC and ST, respectively. Fig. 4-17 plots the values of N_{SC}/K for both SC-TP and SC-SP, which is also the speedup factor of ST over SC in DC analysis. Since ST uses the smallest number of samples, it is more efficient over SC-TP and SC-SP. When low-order generalized polynomial chaos expansions are used ($p \leq 3$), the speedup factor over SC-SP is below 10. The speedup factor can be above 10 if $p \geq 4$, and it gets larger as p increases. In

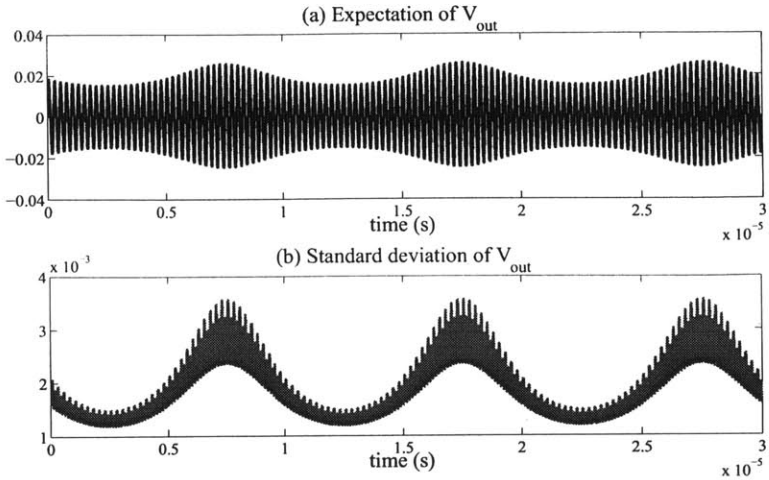


Figure 4-16: Uncertainties of V_{out}=$V_{\text{out1}} - V_{\text{out2}}$ of the double-balanced mixer.

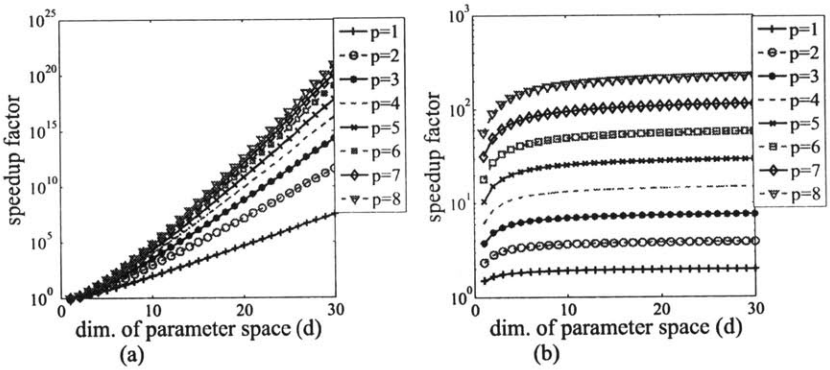


Figure 4-17: The speedup factor of ST over SC caused by sample selection: (a) ST versus SC-TP, (b) ST versus SC-SP. This is also the speedup factor in DC analysis.

high-dimensional cases ($d \gg 1$), the speedup factor of ST over SC-SP only depends on p. It is the similar case if Smolyak sparse grids are used in SC [54]. For example, compared with the sparse-grid SC in [54], our ST has a speedup factor of 2^p if $d \gg 1$.

Transient Simulation: The speedup factor of ST over SC in a transient simulation can be estimated as

$$\nu_{\text{Trans}} \approx (N_{\text{SC}}/K) \times \kappa, \text{ with } \kappa > 1, \tag{4.24}$$

which is larger than ν_{DC}. The first part is the same as in DC analysis. The second

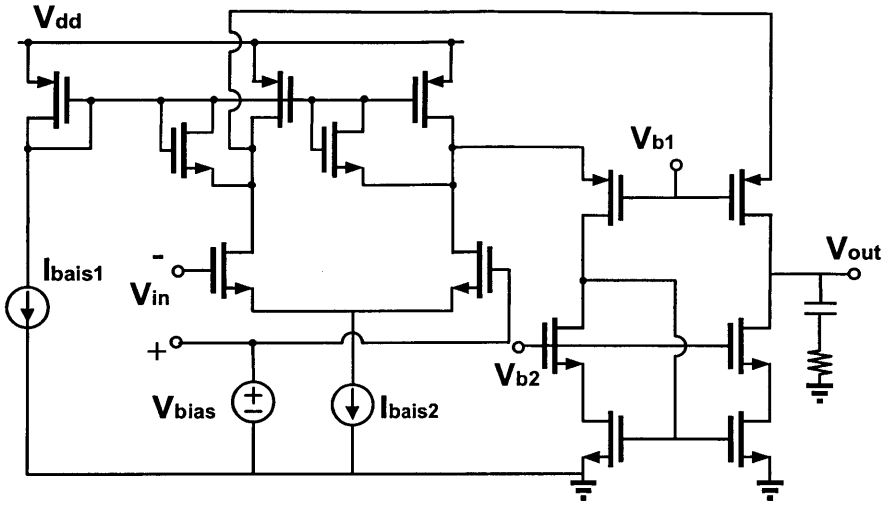


Figure 4-18: The schematic of a CMOS folded-cascode operational amplifier.

part κ represents the speedup caused by adaptive time stepping in our intrusive ST simulator, which is case dependent. For weakly nonlinear analog circuits (e.g., the SC amplifier in Section 4.3.1), κ can be below 10. For digital cells (e.g., the SRAM cell in Section 4.3.3) and multi-rate RF circuits (e.g., the double-balanced mixer in Section 4.3.5), SC-based transient simulation can be prohibitively expensive due to the inefficiency of using a small uniform time step size. In this case, κ can be significantly large.

4.4 Limitations and Possible Solutions

Our proposed simulator has some theoretical limitations.

4.4.1 Discontinuous Solutions

First, the proposed algorithm is only applicable when the solution smoothly depends on the random parameters. This is true for many analog/RF circuits and MEMS problems, but such an assumption may fail for digital circuits or when process variations become too large. For instance, the operational amplifier in Fig. 4-18 is very sensitive to process variations. The static output voltage changes smoothly when the

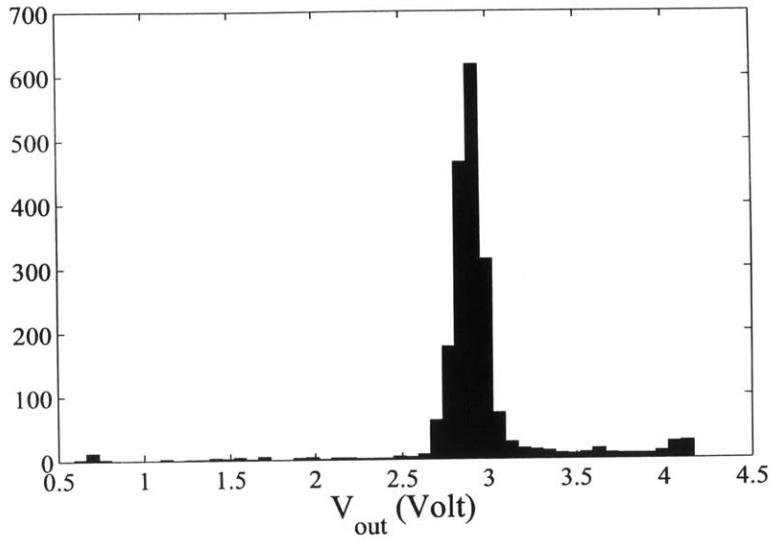


Figure 4-19: Monte-Carlo simulation results of a CMOS operational amplifier.

process variations are very small. However, when we increase the process variations to some extent, the output voltage may suddenly jump from one range to another and the whole circuit does not work in the linear range. Fig. 4-19 shows the histogram of the DC output voltage simulated by Monte Carlo with 2000 samples. Clearly, some output voltages are close to 0, and some approach the supply voltage (4.5 V), implying that the state variables of this circuit are not changing smoothly under process variations.

In order to solve this problem, one possible solution is to first partition the parameter space and then to construct a local approximation for each sub-domain. However, it is not clear how to partition the parameter space in an efficient and accurate way (especially when the parameter space has a high dimension).

4.4.2 Long-Term Integration

The proposed stochastic testing simulator may not work very well if one needs to run a long-term transient simulation. This is because that the variances of waveforms increase as time evolves (as shown in Fig. 4-20, where the waveforms corresponding

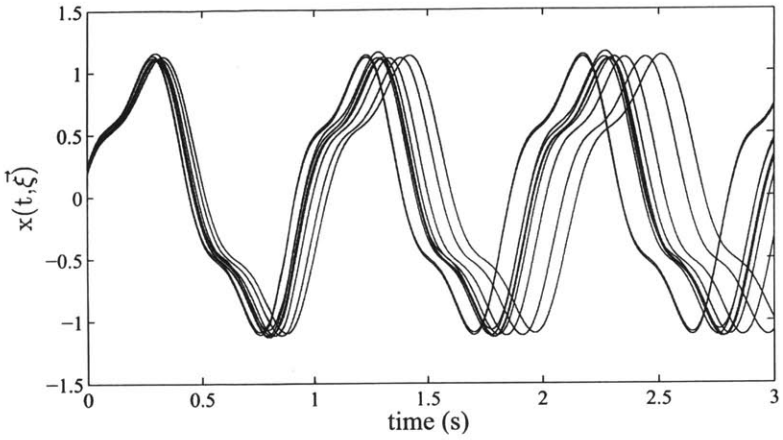


Figure 4-20: The variations of circuit waveforms increase in transient simulation.

to different random parameter realizations are plotted).

This problem may be solved by directly computing the periodic steady states when simulating communication circuits or power electronic circuits (as will be presented in Chapter 5). However, long-term simulation becomes a challenging task when one is interested in the transient behavior instead of a steady state. One possible solution is to develop some novel time-dependent stochastic basis functions to approximate the solutions more accurately.

Chapter 5

Stochastic-Testing Periodic Steady-State Solver

Designers are interested in periodic steady-state analysis when designing analog/RF circuits or power electronic systems [128–134]. Such circuits include both forced (e.g., amplifiers, mixers, power converters) and autonomous cases (also called unforced circuits such as oscillators). Popular periodic steady-state simulation algorithms include shooting Newton, finite difference, harmonic balance, and their variants.

This chapter focuses on the development of uncertainty quantification algorithms for computing the stochastic periodic steady-state solutions caused by process variations. We propose a novel stochastic simulator by combining stochastic testing method with shooting Newton method. Our algorithm can be applied to simulate both forced and autonomous circuits. Extending our ideas to other types of periodic steady-state solvers is straightforward.

The numerical results of our simulator on some analog/RF circuits show remarkable speedup over the stochastic Galerkin approach. For many examples with low-dimensional random parameters, our technique is 2 to 3 orders of magnitude faster than Monte Carlo.

5.1 Review of Shooting Newton Method

In order to show the concepts and numerical solvers for deterministic circuits, we consider a general nonlinear circuit equation without uncertainties:

$$\frac{d\vec{q}(\vec{x}(t))}{dt} + \vec{f}(\vec{x}(t), \vec{u}(t)) = 0. \tag{5.1}$$

We assume that as time involves a periodic steady-state $\vec{x}(t+T) = \vec{x}(t)$ is achieved for any $t > t'$. Many numerical solvers are capable of computing the periodic steady-state solutions [128–134]. In the following, we briefly review shooting Newton method that will be extended to uncertainty analysis. More details on shooting Newton can be found in [128–131].

5.1.1 Forced Circuits

Under a periodic input $\vec{u}(t)$, there exists a periodic steady-state solution $\vec{x}(t) = \vec{x}(t+T)$, where the smallest scalar $T > 0$ is the period known from the input. Shooting Newton method computes $y = \vec{x}(0)$ by solving the Boundary Value Problem (BVP)

$$\vec{\psi}(y) = \vec{\phi}(y, 0, T) - y = 0. \tag{5.2}$$

Here $\vec{\phi}(y, t_0, t)$ is the state transition function, which actually is the state vector $\vec{x}(t + t_0)$ evolving from the initial condition $\vec{x}(t_0) = y$. In order to compute y, Newton's iterations can be applied.

For a general nonlinear dynamic system, there is no analytical form for the state transition function. However, the value of $\vec{\phi}(y, t_0, t)$ can be evaluated numerically: starting from t_0 and using y as an initial condition, performing time-domain integration (i.e., transient simulation) of (5.1) until the time point t, one can obtain the new state vector $\vec{x}(t)$ which is the value of $\vec{\phi}(y, t_0, t)$. Obviously, $\vec{\phi}(y, 0, T) = \vec{x}(T)$ when $y = \vec{x}(0)$.

5.1.2 Oscillator Circuits

For autonomous circuits (i.e., oscillators), $\vec{u}(t) = \vec{u}$ is constant and T is unknown, thus a phase condition must be added. For example, by fixing the j-th element of $\vec{x}(0)$, one uses the boundary value problem

$$\bar{\phi}(y,T) = \begin{bmatrix} \vec{\psi}(y,T) \\ \chi(y) \end{bmatrix} = \begin{bmatrix} \vec{\phi}(y,0,T) - y \\ y_j - \lambda \end{bmatrix} = 0 \tag{5.3}$$

to compute $y = \vec{x}(0)$ and T. Here y_j is the j-th element of y, and λ is a properly selected scalar constant.

5.2 Proposed Stochastic Periodic Steady-State Solver

Let $\mathcal{H}$=$\{H_1(\vec{\xi}), \cdots, H_K(\vec{\xi})\}$ represent the generalized polynomial chaos basis functions with total polynomial order bounded by p, and $\hat{\mathbf{w}} = [\hat{w}_1; \cdots; \hat{w}_K]$ denote the collection of the corresponding coefficients, we define an operator:

$$\mathbb{M}(\mathcal{H}, \hat{\mathbf{w}}, \vec{\xi}) := \tilde{w}(\vec{\xi}) = \sum_{k=1}^{K} \hat{w}_k H_k(\vec{\xi})$$

which converts vector $\hat{\mathbf{w}}$ to a corresponding generalized polynomial chaos approximation $\tilde{w}(\vec{\xi})$. Given a set of testing samples $\mathcal{S} = \{\vec{\xi}^1, \cdots, \vec{\xi}^K\}$, $\mathbf{V} \in \mathbb{R}^{K \times K}$ denotes the Vandermonde-like matrix in (4.8). We still use $\mathbf{W}_n = \mathbf{V} \otimes \mathbf{I}_n$, where $\otimes$ is the Kronecker product operator and $\mathbf{I}_n$ is an identity matrix of size n.

5.2.1 Formulation for Forced Circuits

For a forced circuit, we directly perform uncertainty quantification based on the coupled deterministic DAE (4.4) formed by our stochastic testing simulator. The generalized polynomial-chaos approximated solution can also be written as $\tilde{x}(t, \vec{\xi}) := \mathbb{M}(\mathcal{H}, \hat{\mathbf{x}}(t), \vec{\xi})$ where $\hat{\mathbf{x}}(t) = [\hat{x}_1(t); \cdots; \hat{x}_K(t)] \in \mathbb{R}^{nK}$ collects all generalized polynomial chaos coefficients of $\vec{x}(t, \vec{\xi})$ (as already defined in Chapter 2).

The stochastic state vector $\vec{x}(t,\vec{\xi})$ is periodic for any $\vec{\xi} \in \Omega$ if and only if $\hat{\mathbf{x}}(t)$ is periodic. Therefore, we have

$$\mathbf{g}(\hat{\mathbf{y}}) = \Phi(\hat{\mathbf{y}}, 0, T) - \hat{\mathbf{y}} = 0. \tag{5.4}$$

In this equation, $\hat{\mathbf{y}} = \hat{\mathbf{x}}(0)$, and $\Phi(\hat{\mathbf{y}}, 0, T)$ is the state transition function of (4.4). Similar to the deterministic case, $\Phi(\hat{\mathbf{y}}, 0, T)$ can be computed by a transient simulation of (4.4) for one period T with $\hat{\mathbf{y}}$ as the initial condition at $t = 0$.

5.2.2 Formulation for Autonomous Circuits

For unforced cases, we cannot directly use (4.4) for periodic steady-state analysis since no periodic steady-state solution exists. This is because that the period $T(\vec{\xi})$ is parameter dependent. For each realization of $\vec{\xi}$, $\vec{x}(t,\vec{\xi})$ is periodic. However, when we consider the waveform in the whole parameter space, $\vec{x}(t,\vec{\xi})$ is not periodic.

In order to compute the steady state, we modify (1.3) by scaling the time axis as done in [134]. Let T_0 be the oscillation period for the nominal case, and let $a(\vec{\xi})$ is a parameter-dependent unknown scaling factor, then we write $T(\vec{\xi})$ as

$$T(\vec{\xi}) = T_0 a(\vec{\xi}) \approx T_0 \mathbb{M}(\mathcal{H}, \hat{\mathbf{a}}, \vec{\xi})$$

where $\hat{\mathbf{a}}$=$[\hat{a}_1; \cdots ; \hat{a}_K] \in \mathbb{R}^K$ collects the generalized polynomial chaos coefficients of $a(\vec{\xi})$. Define a new time variable τ such that

$$t = a(\vec{\xi})\tau \approx \mathbb{M}(\mathcal{H}, \hat{\mathbf{a}}, \vec{\xi})\tau,$$

then $\vec{z}(\tau,\vec{\xi}) = \vec{x}(t,\vec{\xi})$ becomes the state vector of the following stochastic differential algebraic equation

$$\frac{d\vec{q}(\vec{z}(\tau,\vec{\xi}),\vec{\xi})}{d\tau} + a(\vec{\xi})\vec{f}(\vec{z}(\tau,\vec{\xi}),\vec{u},\vec{\xi}) = 0. \tag{5.5}$$

Note that $\vec{u}(t) = \vec{u}$ is time-invariant. Replacing $\vec{z}(\tau,\vec{\xi})$ and $a(\vec{\xi})$ in (5.5) with their

generalized polynomial chaos approximations $\tilde{z}(\tau,\vec{\xi})$ and $\tilde{a}(\vec{\xi})$, respectively, and enforcing the resulting residual to zero for any $\vec{\xi_k} \in \mathcal{S}$, we get a deterministic equation

$$\frac{dQ(\hat{\mathbf{z}}(\tau))}{d\tau} + F(\hat{\mathbf{z}}(\tau), \hat{\mathbf{a}}) = 0. \tag{5.6}$$

Here $\hat{\mathbf{z}}(\tau)$=$[\hat{z}_1(\tau); \cdots ; \hat{z}_K(\tau)]$ denotes the generalized polynomial chaos coefficients of $\vec{z}(\tau,\xi)$. The nonlinear functions are decided by

$$Q(\hat{\mathbf{z}}(\tau)) = [\vec{q}_1(\hat{\mathbf{z}}(\tau)); \cdots ; \vec{q}_K(\hat{\mathbf{z}}(\tau))]\,,\ F(\hat{\mathbf{z}}(\tau), \hat{\mathbf{a}}) = \left[\vec{f}_1(\hat{\mathbf{z}}(\tau)); \cdots ; \vec{f}_K(\hat{\mathbf{z}}(\tau))\right]$$

with

$$\vec{q}_k(\hat{\mathbf{z}}(\tau)) = \vec{q}(\tilde{z}(\tau,\vec{\xi^k}),\vec{\xi^k}),\ \vec{f}_k(\hat{\mathbf{z}}(\tau)) = \tilde{a}(\vec{\xi_k})\vec{f}(\tilde{z}(\tau,\vec{\xi_k}),\vec{u},\vec{\xi^k}).$$

Let $\hat{\mathbf{y}} := [\hat{\mathbf{z}}(0); \hat{\mathbf{a}}]$ denote that unknown variables that we want to compute, which includes the generalized polynomial-chaos coefficients for both $\vec{z}(0,\vec{\xi})$ and $a(\vec{\xi})$. By enforcing periodicity of the scaled waveform and by fixing the j-th component of $\vec{z}(0,\vec{\xi})$ at λ, we can set up the following boundary-value problem

$$\mathbf{g}(\hat{\mathbf{y}}) = \begin{bmatrix} \Psi(\hat{\mathbf{z}}(0), \hat{\mathbf{a}}) \\ \chi(\hat{\mathbf{z}}(0)) \end{bmatrix} = \begin{bmatrix} \Phi(\hat{\mathbf{z}}(0), 0, T_0, \hat{\mathbf{a}}) - \hat{\mathbf{z}}(0) \\ \chi(\hat{\mathbf{z}}(0)) \end{bmatrix} = 0. \tag{5.7}$$

Here the state transition function $\Phi(\hat{\mathbf{z}}(0), 0, T_0, \hat{\mathbf{a}})$ for (5.6) depends on $\hat{\mathbf{a}}$, and the phase constraint $\chi(\hat{\mathbf{z}}(0)) = 0 \in \mathbb{R}^K$ is

$$\chi(\hat{\mathbf{z}}(0)) = \left[\hat{\mathbf{z}}_j(0) - \lambda;\ \hat{\mathbf{z}}_{j+n}(0);\ \cdots ;\ \hat{\mathbf{z}}_{j+(K-1)n}(0)\right] = 0,$$

with $\hat{\mathbf{z}}_j(0)$ denotes the j-th component of $\hat{\mathbf{z}}(0)$.

5.3 Numerical Solvers

5.3.1 Coupled Matrix Solver

In order to solve (5.4) and (5.7), we start from an initial guess $\hat{\mathbf{y}}^0$ and use Newton's iteration

$$\text{solve } \Delta\hat{\mathbf{y}} = \mathbf{J}^{-1}(\hat{\mathbf{y}}^j)\mathbf{g}(\hat{\mathbf{y}}^j), \text{ update } \hat{\mathbf{y}}^{j+1} = \hat{\mathbf{y}}^j - \Delta\hat{\mathbf{y}} \tag{5.8}$$

until convergence. The value of function $\mathbf{g}(\hat{\mathbf{y}})$ can be evaluated by running a transient simulation of (4.4) or (5.6) for one period. The main problem is how to evaluate the Jacobian $\mathbf{J}(\hat{\mathbf{y}})$ and how to solve the linear system equation in (5.8).

Forced Case. For a forced case, the Jacobian of (5.4) is

$$\mathbf{J}_{\text{forced}} = \mathbf{M}_{\hat{\mathbf{y}}} - \mathbf{I}, \text{ with } \mathbf{M}_{\hat{\mathbf{y}}} = \frac{\partial\Phi\left(\hat{\mathbf{y}}, 0, T\right)}{\partial\hat{\mathbf{y}}}. \tag{5.9}$$

Here $\mathbf{M}_{\hat{\mathbf{y}}}$ is the Monodromy matrix of (4.4), which can be obtained from linearizations along the trajectory starting from $\hat{\mathbf{x}}(0) = \hat{\mathbf{y}}$ to $\hat{\mathbf{x}}(T)$. This step is the same as the deterministic case detailed in [130] and thus skipped here.

Autonomous Case. The Jacobian of (5.7) reads

$$\mathbf{J}_{\text{osc}} = \begin{bmatrix} \mathbf{J}_{11} & \mathbf{J}_{12} \\ \mathbf{J}_{21} & 0 \end{bmatrix}. \tag{5.10}$$

Submatrix $\mathbf{J}_{11} = \frac{\partial\Psi(\hat{\mathbf{z}}(0),\hat{\mathbf{a}})}{\partial\hat{\mathbf{z}}(0)}$ can be calculated in the same way of computing $\mathbf{J}_{\text{forced}}$; $\mathbf{J}_{21} = \frac{\partial\chi(\hat{\mathbf{z}}(0))}{\partial\hat{\mathbf{z}}(0)}$ is easy to calculate since $\chi(\hat{\mathbf{z}}(0))$ is linear w.r.t. $\hat{\mathbf{z}}(0)$. Submatrix $\mathbf{J}_{12}$ is

$$\mathbf{J}_{12} = \frac{\partial\Psi(\hat{\mathbf{z}}(0),\hat{\mathbf{a}})}{\partial\hat{\mathbf{a}}} = \frac{\partial\Phi(\hat{\mathbf{z}}(0),0,T_0,\hat{\mathbf{a}})}{\partial\hat{\mathbf{a}}} = \frac{\partial\hat{\mathbf{z}}(T_0)}{\partial\hat{\mathbf{a}}}. \tag{5.11}$$

Let $\tau_0 = 0 < \tau_1 < \cdots < \tau_N = T_0$ be a set of discrete time points on the scaled time axis τ, and $h_k = \tau_k - \tau_{k-1}$ be the step size in the transient simulation of (5.6).

We denote the discretized trajectory by $\hat{\mathbf{z}}_{(k)} = \hat{\mathbf{z}}(\tau_k)$. At τ_k, we have

$$Q(\hat{\mathbf{z}}_{(k)}) - Q(\hat{\mathbf{z}}_{(k-1)}) = \left(\gamma_1 F(\hat{\mathbf{z}}_{(k)}, \hat{\mathbf{a}}) + \gamma_2 F(\hat{\mathbf{z}}_{(k-1)}, \hat{\mathbf{a}})\right) h_k$$

with $\gamma_1 = \gamma_2 = 0.5$ for Trapezoidal rule and $\gamma_1 = 1$, $\gamma_2 = 0$ for backward Euler. Taking derivatives on both sides of the above equation yields

$$\frac{\partial \hat{\mathbf{z}}_{(k)}}{\partial \hat{\mathbf{a}}} = (\mathbf{E}_k - \gamma_1 \mathbf{A}_k h_k)^{-1}(\mathbf{E}_{k-1} + \gamma_2 \mathbf{A}_{k-1} h_k)\frac{\partial \hat{\mathbf{z}}_{(k-1)}}{\partial \hat{\mathbf{a}}} + (\mathbf{E}_k - \gamma_1 \mathbf{A}_k h_k)^{-1} h_k(\gamma_1 \mathbf{P}_k + \gamma_2 \mathbf{P}_{k-1}) \tag{5.12}$$

with $\mathbf{E}_k = \frac{\partial Q(\hat{\mathbf{z}}_{(k)})}{\partial \hat{\mathbf{z}}_{(k)}}$, $\mathbf{A}_k = \frac{\partial F(\hat{\mathbf{z}}_{(k)}, \hat{\mathbf{a}})}{\partial \hat{\mathbf{z}}_{(k)}}$ and $\mathbf{P}_k = \frac{\partial F(\hat{\mathbf{z}}_{(k)}, \hat{\mathbf{a}})}{\partial \hat{\mathbf{a}}}$. Starting from $\frac{\partial \hat{\mathbf{z}}_{(0)}}{\partial \hat{\mathbf{a}}} = 0$, one gets $\mathbf{J}_{12} = \frac{\partial \hat{\mathbf{z}}_{(N)}}{\partial \hat{\mathbf{a}}}$ by iterating (5.12).

Similar to the deterministic cases [128–131], the Jacobian is a dense matrix due to the matrix chain operations. Therefore, solving the linear system in (5.8) costs $O(n^3K^3)$ if a direct matrix solver is applied, similar to the cost in [108].

5.3.2 Decoupled Matrix Solver

By properly choosing a transformation matrix $\mathbf{P}$, Equations (5.4) and (5.7) can be converted to

$$\mathbf{P}\mathbf{g}(\hat{\mathbf{y}}) = \begin{bmatrix} \mathbf{g}_1(\tilde{y}(\vec{\xi}^1)) \\ \vdots \\ \mathbf{g}_K(\tilde{y}(\vec{\xi}^K)) \end{bmatrix}, \text{ with } \begin{bmatrix} \tilde{y}(\vec{\xi}^1) \\ \vdots \\ \tilde{y}(\vec{\xi}^K) \end{bmatrix} = \mathbf{P}\hat{\mathbf{y}}. \tag{5.13}$$

Consequently, the Jacobian in (5.8) can be rewritten as

$$\mathbf{J}(\hat{\mathbf{y}}) = \mathbf{P}^{-1} \begin{bmatrix} \mathrm{J}_1 & & \\ & \ddots & \\ & & \mathrm{J}_K \end{bmatrix} \mathbf{P}, \text{ with } \mathrm{J}_k = \frac{\partial \mathbf{g}_k(\tilde{y}(\vec{\xi}^k))}{\partial \tilde{y}(\vec{\xi}^k)}. \tag{5.14}$$

Now we proceed to discuss how to calculate each block in (5.14).

Forced Case. We set $\mathbf{P}=\mathbf{W}_n$ and $\tilde{y}(\vec{\xi}^k)=\tilde{x}(0,\vec{\xi}^k)$, then

$$\mathbf{g}_k\left(\tilde{y}(\vec{\xi}^k)\right) = \vec{\phi}\left(\tilde{x}(0,\vec{\xi}^k),0,T\right) - \tilde{x}(0,\vec{\xi}^k) = 0 \tag{5.15}$$

is a deterministic boundary-value problem used to compute the periodic steady state of (1.3), with $\vec{\xi}$ fixed at $\vec{\xi}^k$. In Equation (5.15), $\tilde{x}(0,\vec{\xi}^k) \in \mathbb{R}^n$ is unknown, $\tilde{x}(t,\vec{\xi}^k) = \vec{\phi}\left(\tilde{x}(0,\vec{\xi}^k),0,t\right)$ is the state transition function, and J_k can be formed using existing techniques based on numerical time-domain integration such as a backward Euler or trapezoidal rule [129].

Autonomous Case. Let $\tilde{y}(\vec{\xi}^k) = [\tilde{z}(0,\vec{\xi}^k);\tilde{a}(\xi^k)]$, and $\mathbf{P} = \mathbf{W}_{n+1}\Theta$ where Θ is a proper permutation matrix, then

$$\mathbf{g}_k\left(\tilde{y}(\vec{\xi}^k)\right) = \begin{bmatrix} \vec{\phi}\left(\tilde{z}(0,\vec{\xi}^k),0,T_0,\tilde{a}(\vec{\xi}^k)\right) - \tilde{z}(0,\vec{\xi}^k) \\ \tilde{z}_j(0,\vec{\xi}^k) \end{bmatrix} = 0$$

is a deterministic boundary-value problem used to compute the periodic steady state of (5.5), with the parameter $\vec{\xi}$ fixed at $\vec{\xi}^k$. Here $\tilde{z}(0,\vec{\xi}^k)$ and $\tilde{a}(\vec{\xi}^k)$ are the unknowns, and $\tilde{z}(\tau,\vec{\xi}^k) = \vec{\phi}\left(\tilde{z}(0,\vec{\xi}^k),0,\tau,\tilde{a}(\vec{\xi}^k)\right)$ is a state transition function dependent on $a(\vec{\xi}) = \tilde{a}(\vec{\xi}^k)$. The small Jacobian J_k can also be formed by existing numerical techniques [131, 134].

Intrusive Solver. We directly compute the generalized polynomial chaos coefficients by solving (5.4) or (5.7), with decoupling **inside** the Newton's iterations (5.8). Specifically, inside each iteration, Eq. (4.4) or (5.6) is first integrated for one period, and the state trajectories are converted to the generalized polynomial chaos approximations [i.e., $\tilde{x}(t,\vec{\xi}^k)$'s in forced circuits, or $\tilde{z}(\tau,\vec{\xi}^k)$'s and $\tilde{a}(\vec{\xi}^k)$'s in unforced circuits]. Then J_k's are formed as done in existing deterministic periodic steady-state solvers [128–131]. Finally, based on (5.14) each small block is solved independently to update $\hat{\mathbf{y}}^j$. Doing so allows simulating (4.4) or (5.6) with flexible time stepping controls inside the intrusive transient solver [34], such that all components of $\hat{\mathbf{x}}(t)$ [or $\hat{\mathbf{z}}(\tau)$] are located on the same adaptive time grid. This allows us to directly extract the statistical information of the time-domain waveforms and other performance metrics

(e.g., statistical transient power).

Complexity. Since $\Theta^{-1}=\Theta^T$, $\mathbf{W}_n^{-1}=\mathbf{V}^{-1}\otimes\mathbf{I}_n$ and $\mathbf{V}^{-1}$ can be easily computed [34], the cost of decoupling in (5.14) is negligible. After decoupling, one can solve each small linear system equation as done in deterministic periodic steady-state solvers [128–131]. The total cost is $O(Kn^3)$ if a direct matrix solver is used. For large-scale circuits, one can use matrix-free iterative methods [130] at the cost of $O(Kn^\beta)$ where β is normally 1.5$\sim$2. This intrusive decoupled solver could be easily parallelized potentially leading to further speedup.

5.4 Numerical Results

Our algorithm was implemented in a Matlab circuit simulator. All experiments were run on a workstation with 3.3GHz 4-GB RAM.

5.4.1 Low-Noise Amplifier (LNA)

The LNA in Fig. 4-7 is used as an example of forced circuits. The ratios of the transistors are W_1/L_1=W_2/L_2=500/0.35 and W_3/L_3=50/0.35. The design parameters are: $V_{\rm dd}$=1.5 V, R_1=50Ω, R_2=2 kΩ, C_1=10 pF, C_L=0.5 pF, L_1=20 nH and L_3=7 nH. We introduce four random parameters. Temperature T=300 + $\mathcal{N}(0,1600)$ K is a Gaussian variable influencing transistor threshold voltage; R_3=1 + $\mathcal{U}(-0.1,0.1)$ kΩ and L_2=1.4 + $\mathcal{U}(0.6,0.6)$ nH have uniform distributions; the threshold voltage under zero $V_{\rm bs}$ is $V_{\rm T}$=0.4238 + $\mathcal{N}(0,0.01)$ V. The input is $V_{\rm in} = 0.1\sin(4\pi \times 10^8 t)$ V.

In our stochastic testing-based periodic steady-state solver, an order-3 generalized polynomial chaos expansion (with 35 basis functions) are used to represent the state variables. The computed generalized polynomial chaos coefficients are then used to extract statistical information at a negligible cost. The means and standard deviations (s.t.d) of $V_{\rm out}$ and $I(V_{\rm dd})$ (current from $V_{\rm dd}$) are plotted in Fig. 5-1. Using standard MC, 8000 samples are required to achieve the similar level of accuracy (<1% relative errors for the mean and standard deviation). Fig. 5-2 plots the probability density functions (PDF) of the total harmonic distortion (THD) and power consumption from

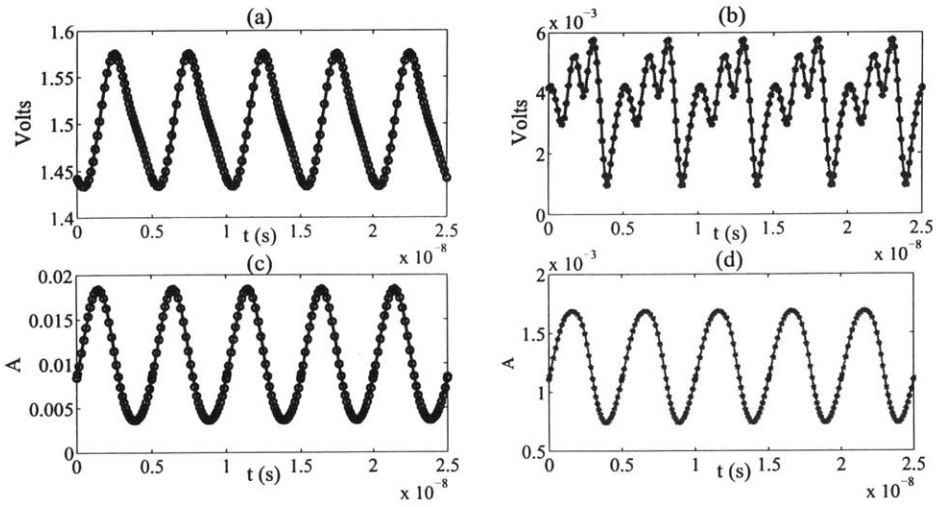


Figure 5-1: Periodic steady-state waveforms for the LNA. (a) & (b): mean and s.t.d of V_{out}; (c) & (d): mean and s.t.d of $I(V_{\text{dd}})$.

our proposed periodic steady-state solver and MC, respectively. The PDFs from both methods are graphically indistinguishable. The total cost of our decoupled stochastic testing solver is 3.4 seconds, which is 42× faster over the coupled stochastic testing solver, 71× faster over the stochastic Galerkin-based periodic steady-state solver, and 220× faster over MC.

5.4.2 BJT Colpitts Oscillator

The BJT Colpitts oscillator in Fig. 5-3 is a typical example of autonomous circuits. The design parameters of this circuit are R_1=2.2 kΩ, R_2=R_3=10 kΩ, C_2=100 pF, C_3=0.1μF, and α=0.992 for the BJT. The oscillation frequency is mainly determined by L_1, C_1 and C_2. We assume that L_1=$150 + \mathcal{N}(0, 9)$ nH and C_1=$100 + \mathcal{U}(-10, 10)$ pF are random variables with Gaussian and uniform distributions, respectively.

Setting the generalized polynomial chaos order to 3, the results from our proposed solver and the stochastic Galerkin-based solver [108] are indistinguishable. Fig. 5-4 shows some realizations of V_{out} obtained by our solver. The variation looks small on the scaled time axis, but it is significant on the original time axis due to the uncertainties of the oscillation frequency. The CPU time of our decoupled stochas-

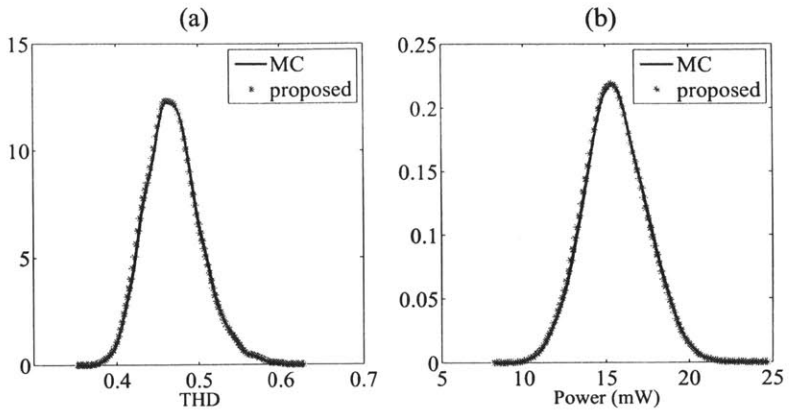


Figure 5-2: Probability density functions. (a)total harmonic distortion and (b) power dissipation.

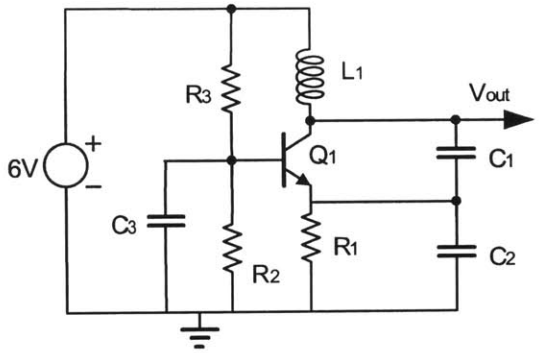


Figure 5-3: Schematic of the BJT Colpitts oscillator.

tic testing-based solver is 4.9 seconds, which is 2× and 5× faster over the coupled stochastic testing-based solver and the stochastic Galerkin-based solver [108], respectively.

Finally, our solver is compared with standard MC. The computed mean and standard deviation (both in nanosecond) of the oscillation period are shown in Table 5.1. In order to achieve the similar level of accuracy, MC must use 5000 samples, which is about 507× slower than using our stochastic testing-based simulator. The distributions of the oscillation period from both methods are consistent, as shown in Fig. 5-5.

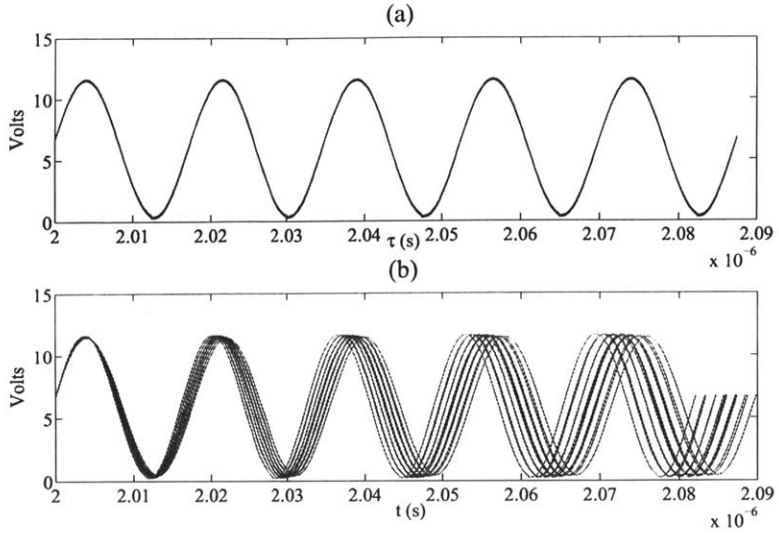


Figure 5-4: Realizations of V_{out} for the Colpitts oscillator. (a) on the scaled time axis, (b) on the original time axis.

Table 5.1: Simulation results of the oscillation period by our proposed method and Monte Carlo.

	Monte Carlo			Proposed
# samples	500	2000	5000	10
mean value (ns)	17.194	17.203	17.205	17.205
s.t.d value (ns)	2.995	3.018	3.026	3.028
CPU time (s)	252	1013	2486	4.9

5.4.3 Accuracy and Efficiency

We increased the generalized polynomial chaos order from 1 to 6, and treated the results from the 6th-order generalized polynomial chaos expansion as the "exact" solutions. Fig. 5-6 plots the relative errors of $\hat{\mathbf{y}}$ and the speedup factors caused by decoupling. The errors rapidly reduce to below 10^{-4}, and the convergence slows down when the errors approach 10^{-5}, i.e., the threshold for the Newton's iterations which dominates the accuracy. In Fig. 5-6(b), the speedup curve for the LNA has the same slope as K^2 on a logarithmic scale, implying an $O(K^2)$ speedup caused by decoupling. The speedup for the Colpitts oscillator is however not significant, since device evaluations dominate the total cost for this small circuit. Generally, the $O(K^2)$ speedup is more obvious for large-scale circuits.

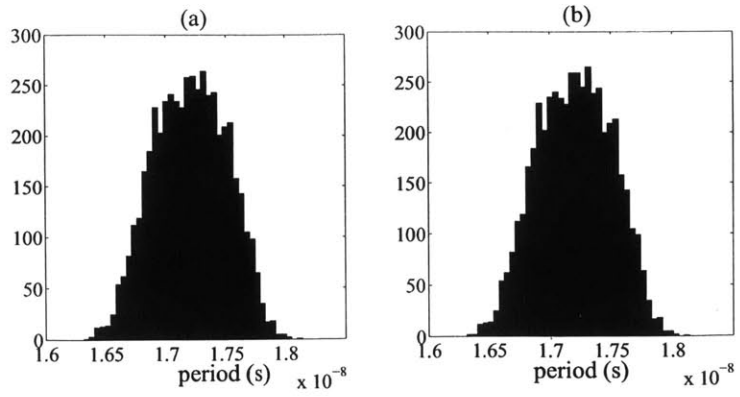


Figure 5-5: Distributions of the oscillation period: (a) from our proposed method, (b) from MC (5000 samples).

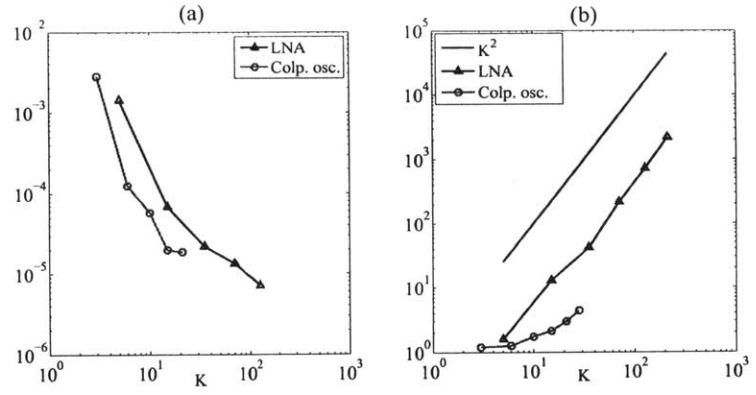


Figure 5-6: (a) Relative error of our solver. (b) Speedup factor caused by decoupling.

5.5 Limitations and Possible Solutions

5.5.1 Failure of Stochastic Periodic Steady-State Solvers

Our simulator computes the scaled periodic steady-state solutions by shooting Newton. This algorithm, however, requires an initial guess that is close to the exact solution. For circuits with small variations, one may utilize the simulation result of a nominal circuit as an initial guess. However, this method does not work well when the variations become very large.

One may develop a frequency-domain simulator (e.g., using harmonic balance technique) to solve the above problem. Harmonic balance may converge to a static

equilibrium point, and thus a continuation method may also help to improve the robustness. When the variation is not that large, it is worth trying to provide a better initial guess for shooting Newton using sensitivity analysis [133].

5.5.2 Simulating High-Q Oscillators

Our simulator does not work well for high-quality (i.e., high-Q) oscillators. For deterministic cases, high-Q circuits can be solved by employing envelope following techniques [135–137]. It would be great if these algorithms can be extended to the stochastic cases. Unfortunately, this is not a trivial task because the waveform envelopes, quality factors and oscillation frequencies are all dependent on process variations.

Chapter 6

High-Dimensional Hierarchical Uncertainty Quantification

Since many electronic systems are designed in a hierarchical way, this chapter exploits such structure and simulate a complex circuit or system by hierarchical uncertainty quantification [37, 39]. Specifically, we first utilize stochastic spectral methods to extract surrogate models for each subsystem. Then, the circuit equations describing the interconnection of all subsystems are solved with stochastic spectral methods again by treating each subsystem as a single random parameter. Typical application examples include (but are not limited to) analog/mixed-signal systems (e.g., phase-lock loops) and MEMS/IC co-design.

The advantages of this approach are two-fold:

- The parameter dimensionality of the whole system can be significantly reduced at the high-level simulation.
- Details of each subsystem can be ignored, leading to fewer unknown variables when simulating the whole system.

This chapter first presents a hierarchical approach by assuming that each subsystem is well described by a low-dimensional generalized polynomial-chaos expansion. Then, we propose some efficient algorithms to accelerate the computation when the parameter dimensionality is very high. The simulation results on a 9-parameter

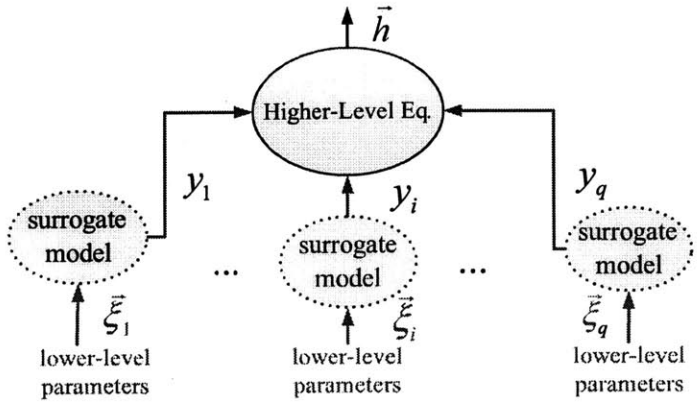


Figure 6-1: Demonstration of hierarchical uncertainty quantification.

MEMS/IC co-design show that our solver is 250× faster over the state-of-the-art solver. Compared with Monte Carlo, the speedup factor of our technique on a high-dimensional MEMS/IC co-design example (with 184 random parameters) is about 100×.

6.1 Hierarchical Uncertainty Quantification

6.1.1 Description of the Basic Idea

Consider Fig. 6-1, where a complex electronic circuit or system has q subsystems. The output y_i of a subsystem is influenced by some bottom-level random parameters $\vec{\xi}_i \in \mathbb{R}^{d_i}$, and the output $\vec{h}$ of the whole system depends on all random parameters $\vec{\xi}_i$'s. In a statistical behavior-level simulator [1], y_i is the performance metric of a small circuit block (e.g., the frequency of a voltage-controlled oscillator) affected by some device-level parameters $\vec{\xi}_i$. Typical surrogate models include linear (quadratic) response surface models [1–3, 138–140], truncated generalized polynomial chaos representations [34, 36], smooth or non-smooth functions, stochastic reduced-order models [105, 141, 142], and some numerical packages that can rapidly evaluate $f_i(\vec{\xi}_i)$ (e.g., computer codes that implement a compact statistical device model).

For simplicity, we assume that y_i only depends on $\vec{\xi}_i$ and does not change with time

or frequency. Directly simulating the whole system can be expensive due to the large problem size and high parameter dimensionality. If y_i's are mutually independent and smoothly dependent on $\vec{\xi_i}$'s, we can accelerate the simulation in a hierarchical way [39]:

- **Step 1**. We use our fast stochastic spectral simulator [34, 36] to extract a surrogate model for each block

$$y_i = g_i(\vec{\xi_i}), \text{ with } \vec{\xi_i} \in \mathbb{R}^{d_i}, \ i = 1, \cdots, q. \tag{6.1}$$

  With the surrogate models, y_i can be evaluated very rapidly. Note that other techniques [1, 88, 141] can also be utilized to build surrogate models. For numerical stability, we define ζ_i by shifting and scaling y_i such that ζ_i has a zero mean and unit variance.

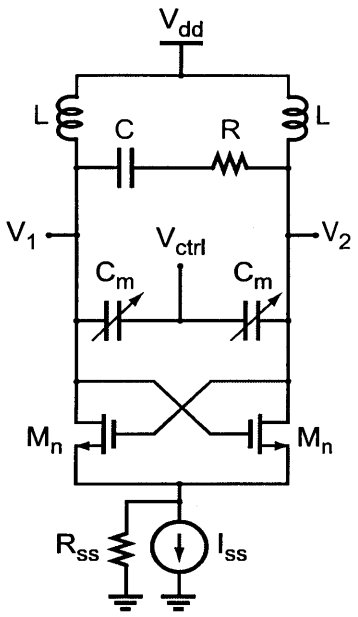
- **Step 2**. By treating ζ_i's as the new random sources, we compute $\vec{h}$ by solving the system-level equation

$$\mathbb{G}(\vec{h}, \vec{\zeta}) = 0, \text{ with } \vec{\zeta} = [\zeta_1, \cdots, \zeta_q], \tag{6.2}$$

  where $\mathbb{G}$ is the abstraction of a proper mathematical operator that describes the interconnections of all subsystems. Again, we use the stochastic testing algorithm [34–36] to solve efficiently this system-level stochastic problem. Stochastic Galerkin and stochastic collocation can be utilized as well. Note that (6.2) can be either an algebraic or a differential equation, depending on the specific problems.

6.1.2 Numerical Implementation

The main challenge of our hierarchical uncertainty quantification flow lies in Step 2. In order to employ stochastic testing (or other stochastic spectral methods), we need the univariate generalized polynomial basis functions and Gauss quadrature rule of ζ_i, which are not readily available.



Parameter	Value
V_{dd}	2.5V
V_{ctrl}	0-2.5V
C	3.3pF
R	110Ω
L	2.2μH
$(W/L)_n$	4/0.25
I_{ss}	0.5mA
R_{ss}	$10^6\Omega$

Figure 6-2: Schematic of a voltage-control oscillator with MEMS capacitors.

Let $\rho(\zeta_i)$ be the probability density function of ζ_i, then we first construct $p+1$ orthogonal polynomials $\pi_j(\zeta_i)$ via [46] (as detailed in Chapter 2.1.1) and then a set of Gauss quadrature points (c.f. Chapter 2.2.1). The main difficulty is to calculate the recurrence parameters

$$\gamma_j = \frac{\int_{\mathbb{R}} \zeta_i \pi_j^2(\zeta_i)\rho(\zeta_i)d\zeta_i}{\int_{\mathbb{R}} \pi_j^2(\zeta_i)\rho(\zeta_i)d\zeta_i}, \quad \kappa_{j+1} = \frac{\int_{\mathbb{R}} \pi_{j+1}^2(\zeta_i)\rho(\zeta_i)d\zeta_i}{\int_{\mathbb{R}} \pi_j^2(\zeta_i)\rho(\zeta_i)d\zeta_i} \tag{6.3}$$

with $\kappa_0 = 1$. Here $\pi_j(\zeta_i)$ is a degree-j polynomial with leading coefficient 1 generated according to the iterations in (2.3).

It becomes obvious that both the basis functions and quadrature points/weights depend on the probability density function of ζ_i. Unfortunately, unlike the bottom-level random parameters $\vec{\xi}_i$'s that are well defined by process cards, the intermediate-level random parameter ζ_i does not have a given density function. Therefore, the iteration parameters γ_j and κ_j are not known.

When $f_i(\vec{\xi}_i)$ is smooth enough and $\vec{\xi}_i$ is of low dimensionality, we compute the integrals in (6.3) in the parameter space of $\vec{\xi}_i$. In this case, the multi-dimensional

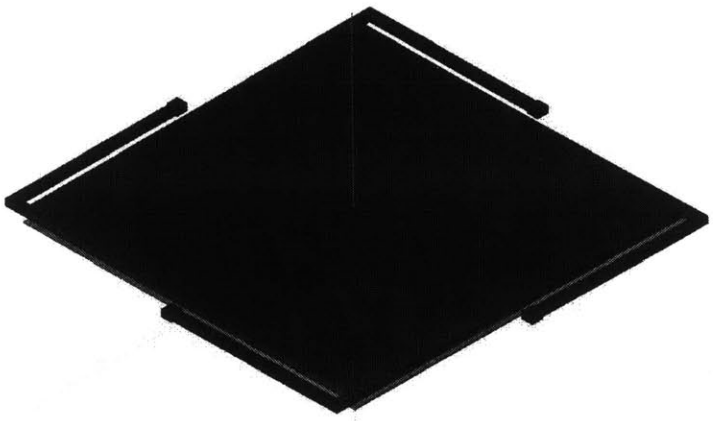


Figure 6-3: Schematic of the MEMS capacitor.

quadrature rule of $\vec{\xi}_i$ is utilized to evaluate the integral (as detailed in Chapter 2.2.2).

6.1.3 A Demonstrative Low-Dimensional MEMS/IC Co-Design Example

As a demonstration, we consider the voltage-controlled oscillator in Fig. 6-2. This oscillator has two independent identical MEMS capacitors C_m, the 3-D schematic of which is shown in Fig. 6-3. Each MEMS capacitor is influenced by four Gaussian-type process and geometric parameters, and the transistor threshold voltage is also influenced by the Gaussian-type temperature variation. Therefore, this circuit has nine random parameters in total. Since it is inefficient to directly solve the coupled stochastic circuit and MEMS equations, our proposed hierarchical stochastic simulator is employed.

Surrogate Model Extraction. The stochastic testing algorithm has been implemented in the commercial MEMS simulator MEMS+ [143] to solve the stochastic MEMS equation (1.4). A 3rd-order generalized polynomial-chaos expansion and 35 testing points are used to calculate the displacements, which then provide the capacitance as a surrogate model. Fig. 6-4 plots the density functions of the MEMS capacitor from our simulator and from Monte Carlo using 1000 samples. The results

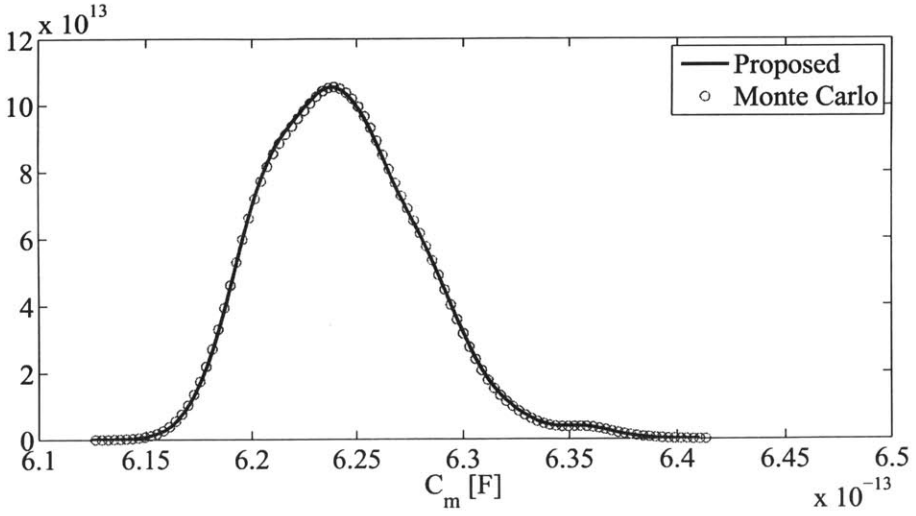


Figure 6-4: Computed probability density function of MEMS capacitor C_m.

match perfectly, and our simulator is about 30× faster.

Higher-Level Simulation The obtained MEMS capacitor models are normalized (and denoted as ζ_1 and ζ_2) such that they have zero means and unit variances. A higher-level equation is constructed, which is the stochastic differential algebraic equation in (1.3) for this example. The constructed basis functions and Gauss quadrature points/weights for ζ_1 are plotted in Fig. 6-5. The stochastic-testing-based periodic steady-state solver [36] is utilized to solve this higher-level stochastic equation to provide 3rd-order generalized polynomial expansions for all branch currents, nodal voltages and the oscillation period. In Fig. 6-6, the computed oscillator period from our hierarchical stochastic spectral simulator is compared with that from the hierarchical Monte Carlo approach [1]. Our approach requires only 20 samples and less than 1 minute for the higher-level stochastic simulation, whereas the method in [1] requires 5000 samples to achieve the similar level of accuracy. Therefore, the speedup factor of our technique is about 250×.

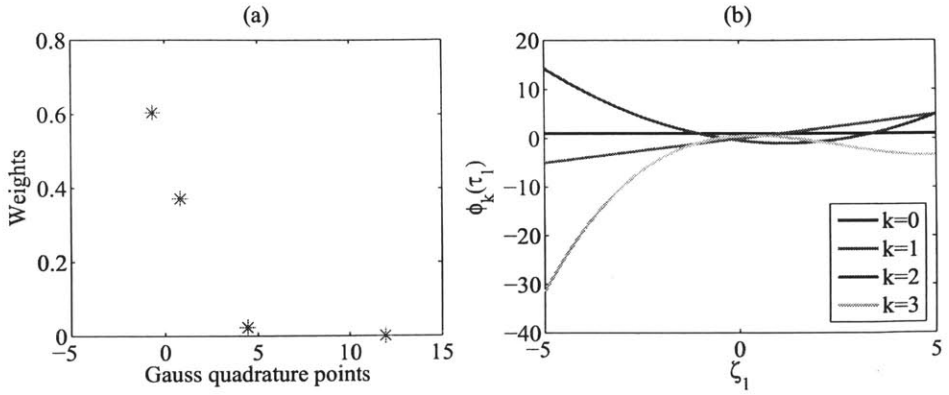


Figure 6-5: The computed Gauss quadrature points/weights and basis functions for the intermediate-level parameter ζ_1.

6.1.4 Challenges in High Dimension

When d_i is large, it is non-trivial to implement hierarchical uncertainty quantification due to the following reasons.

- First, it is non-trivial to obtain a generalized polynomial chaos expansion for y_i, since a huge number of basis functions and samples are required to obtain a good approximation of $y_i(\vec{\xi_i})$.
- Second, when high accuracy is required, it is expensive to implement (6.3) due to the non-trivial integrals when computing κ_j and γ_j. Since the density function of ζ_i is unknown, the integrals must be evaluated in the domain of $\vec{\xi_i}$, with a cost growing exponentially with d_i when a deterministic quadrature rule is used.

In the following two sections, efficient algorithms will be proposed to mitigate the above two problems.

6.2 ANOVA-Based Surrogate Model Extraction

In order to accelerate the low-level simulation, this section develops a sparse stochastic circuit/MEMS simulator based on anchored ANOVA (analysis of variance). Without of loss of generality, let $y = g(\vec{\xi})$ denote the output of a subsystem. We assume that y

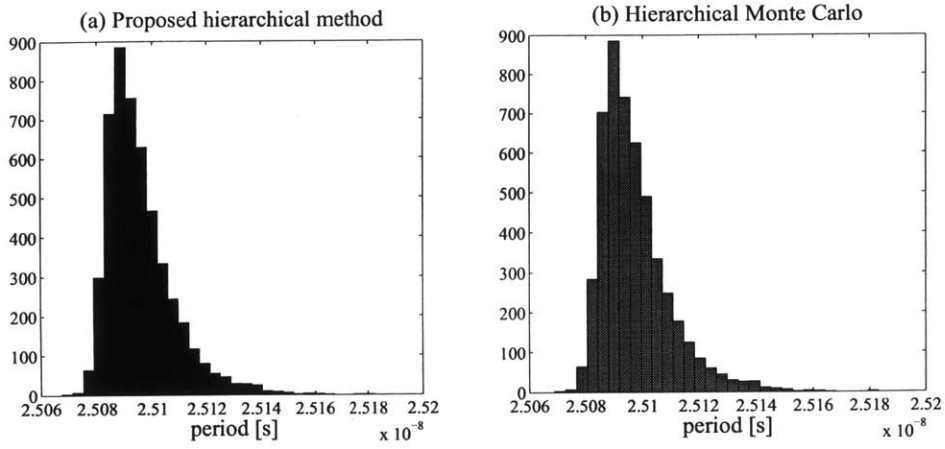


Figure 6-6: Histograms of the oscillator period, (a) from our hierarchical stochastic spectral simulator, (b) from hierarchical Monte Carlo [1].

is a smooth function of the random parameters $\vec{\xi} \in \Omega \subseteq \mathbb{R}^d$ that describe the process variations.

6.2.1 ANOVA and Anchored ANOVA Decomposition

ANOVA. With ANOVA decomposition [144, 145], y can be written as

$$y = g(\vec{\xi}) = \sum_{s \subseteq \mathcal{I}} g_s(\vec{\xi}_s), \tag{6.4}$$

where s is a subset of the full index set $\mathcal{I} = \{1, 2, \cdots, d\}$. Let $\bar{s}$ be the complementary set of s such that $s \cup \bar{s} = \mathcal{I}$ and $s \cap \bar{s} = \emptyset$, and let $|s|$ be the number of elements in s. When $s = \{i_1, \cdots, i_{|s|}\} \neq \emptyset$, we set $\Omega_s = \Omega_{i_1} \otimes \cdots \otimes \Omega_{i_{|s|}}$, $\vec{\xi}_s = [\xi_{i_1}, \cdots, \xi_{i_{|s|}}] \in \Omega_s$ and have the Lebesgue measure

$$d\mu(\vec{\xi}_{\bar{s}}) = \prod_{k \in \bar{s}} (\rho_k(\xi_k)\, d\xi_k). \tag{6.5}$$

Then, $g_s(\vec{\xi}_s)$ in ANOVA decomposition (6.4) is defined recursively by the following formula

$$g_s(\vec{\xi}_s) = \begin{cases} \mathbb{E}\left(g(\vec{\xi})\right) = \int_{\Omega} g(\vec{\xi}) d\mu(\vec{\xi}) = g_0, \text{ if } s = \emptyset \\ \hat{g}_s(\vec{\xi}_s) - \sum_{t \subset s} g_t(\vec{\xi}_t) \ , \text{ if } s \neq \emptyset. \end{cases} \tag{6.6}$$

Here $\mathbb{E}$ is the expectation operator, $\hat{g}_s(\vec{\xi}_s) = \int_{\Omega_{\bar{s}}} g(\vec{\xi}) d\mu(\vec{\xi}_{\bar{s}})$, and the integration is computed for all elements except those in $\vec{\xi}_s$. From (6.6), we have the following intuitive results:

- g_0 is a constant term;
- if s=$\{j\}$, then $\hat{g}_s(\vec{\xi}_s) = \hat{g}_{\{j\}}(\xi_j)$, $g_s(\vec{\xi}_s) = g_{\{j\}}(\xi_j) = \hat{g}_{\{j\}}(\xi_j) - g_0$;
- if s=$\{j,k\}$ and $j < k$, then $\hat{g}_s(\vec{\xi}_s) = \hat{g}_{\{j,k\}}(\xi_j, \xi_k)$ and $g_s(\vec{\xi}_s) = \hat{g}_{\{j,k\}}(\xi_j, \xi_k) - g_{\{j\}}(\xi_j) - g_{\{k\}}(\xi_k) - g_0$;
- both $\hat{g}_s(\vec{\xi}_s)$ and $g_s(\vec{\xi}_s)$ are $|s|$-variable functions, and the decomposition (6.4) has 2^d terms in total.

Example 1. *Consider* $y = g(\vec{\xi}) = g(\xi_1, \xi_2)$. *Since* $\mathcal{I} = \{1,2\}$, *its subset includes* $\emptyset$, $\{1\}$, $\{2\}$ *and* $\{1,2\}$. *As a result, there exist four terms in the ANOVA decomposition (6.4):*

- *for* $s = \emptyset$, $g_\emptyset(\vec{\xi}_\emptyset) = \mathbb{E}\left(g(\vec{\xi})\right) = g_0$ *is a constant;*
- *for* $s = \{1\}$, $g_{\{1\}}(\xi_1)$=$\hat{g}_{\{1\}}(\xi_1) - g_0$, *and* $\hat{g}_{\{1\}}(\xi_1) = \int_{\Omega_2} g(\vec{\xi}) \rho_2(\xi_2) d\xi_2$ *is a univariate function of* ξ_1;
- *for* $s = \{2\}$, $g_{\{2\}}(\xi_2)$=$\hat{g}_{\{2\}}(\xi_2) - g_0$, *and* $\hat{g}_{\{2\}}(\xi_2) = \int_{\Omega_1} g(\vec{\xi}) \rho_1(\xi_1) d\xi_1$ *is a univariate function of* ξ_2;
- *for* s=$\{1,2\}$, $g_{\{1,2\}}(\xi_1, \xi_2)$=$\hat{g}_{\{1,2\}}(\xi_1, \xi_2) - g_{\{1\}}(\xi_1) - g_{\{2\}}(\xi_2) - g_0$. *Since* $\bar{s}$=$\emptyset$, *we have* $\hat{g}_{\{1,2\}}(\xi_1, \xi_2) = g(\vec{\xi})$, *which is a bi-variate function.*

Since all terms in the ANOVA decomposition are mutually orthogonal [144, 145], we have

$$\mathbf{Var}\left(g(\vec{\xi})\right) = \sum_{s \subseteq \mathcal{I}} \mathbf{Var}\left(g_s(\vec{\xi}_s)\right) \tag{6.7}$$

where $\mathbf{Var}(\bullet)$ denotes the variance over the whole parameter space Ω. What makes ANOVA practically useful is that for many engineering problems, $g(\vec{\xi})$ is mainly influenced by the terms that depend only on a small number of variables, and thus it can be well approximated by a truncated ANOVA decomposition

$$g(\vec{\xi}) \approx \sum_{|s| \leq d_{\text{eff}}} g_s(\vec{\xi}_s),\ s \subseteq \mathcal{I} \tag{6.8}$$

where $d_{\text{eff}} \ll d$ is called the **effective dimension**.

Example 2. *Consider* $y = g(\vec{\xi})$ *with* $d = 20$. *In the full ANOVA decomposition (6.4), we need to compute over* 10^6 *terms, which is prohibitively expensive. However, if we set* $d_{\text{eff}} = 2$, *we have the following approximation*

$$g(\vec{\xi}) \approx g_0 + \sum_{j=1}^{20} g_j(\xi_j) + \sum_{1 \leq j < k \leq 20} g_{j,k}(\xi_j, \xi_k) \tag{6.9}$$

which contains only 221 *terms.*

Unfortunately, it is still expensive to obtain the truncated ANOVA decomposition (6.8) due to two reasons. First, the high-dimensional integrals in (6.6) are expensive to compute. Second, the truncated ANOVA decomposition (6.8) still contains lots of terms when d is large. In the following, we introduce anchored ANOVA that solves the first problem. The second issue will be addressed in Section 6.2.2.

Anchored ANOVA. In order to avoid the expensive multidimensional integral computation, [145] has proposed an efficient algorithm which is called anchored ANOVA in [146–148]. Assuming that ξ_k's have standard uniform distributions, anchored ANOVA first chooses a deterministic point called anchored point $\vec{q} = [q_1, \cdots, q_d] \in$

$[0,1]^d$, and then replaces the Lebesgue measure with the Dirac measure

$$d\mu(\vec{\xi_{\bar{s}}}) = \prod_{k\in\bar{s}} (\delta(\xi_k - q_k)\, d\xi_k). \tag{6.10}$$

As a result, $g_0 = g(\vec{q})$, and

$$\hat{g}_s(\vec{\xi_s}) = g\left(\tilde{\xi}\right), \text{ with } \tilde{\xi}_k = \begin{cases} q_k, \text{ if } k \in \bar{s} \\ \xi_k, \text{ otherwise.} \end{cases} \tag{6.11}$$

Here $\tilde{\xi}_k$ denotes the k-th element of $\tilde{\xi} \in \mathbb{R}^d$, q_k is a fixed deterministic value, and ξ_k is a random variable. Anchored ANOVA was further extended to Gaussian random parameters in [147]. In [146, 148, 149], this algorithm was combined with stochastic collocation to efficiently solve high-dimensional stochastic partial differential equations.

Example 3. *Consider $y=g(\xi_1,\xi_2)$. With an anchored point $\vec{q} = [q_1, q_2]$, we have $g_0 = g(q_1,q_2)$, $\hat{g}_{\{1\}}(\xi_1) = g(\xi_1,q_2)$, $\hat{g}_{\{2\}}(\xi_2) = g(q_1,\xi_2)$ and $\hat{g}_{\{1,2\}}(\xi_1,\xi_2) = g(\xi_1,\xi_2)$. Computing these quantities does not involve any high-dimensional integrations.*

6.2.2 Adaptive Anchored ANOVA for Circuit/MEMS Problems

Extension to General Cases. In many circuit and MEMS problems, the process variations can be non-uniform and non-Gaussian. We show that anchored ANOVA can be applied to such general cases.

Observation: The anchored ANOVA in [145] can be applied if $\rho_k(\xi_k) > 0$ for any $\xi_k \in \Omega_k$.

Proof. Let u_k denote the cumulative density function for ξ_k, then u_k can be treated as a random variable uniformly distributed on $[0,1]$. Since $\rho_k(\xi_k) > 0$ for any $\xi_k \in \Omega_k$, there exists $\xi_k = \lambda_k(u_k)$ which maps u_k to ξ_k. Therefore, $g(\xi_1, \cdots, \xi_d) =$

$g\left(\lambda_1(u_1), \cdots, \lambda_d(u_d)\right) = \psi(\vec{u})$ with $\vec{u} = [u_1, \cdots, u_d]$. Following (6.11), we have

$$\hat{\psi}_s(\vec{u}_s) = \psi\left(\tilde{u}\right), \text{ with } \tilde{u}_k = \begin{cases} p_k, \text{ if } k \in \bar{s} \\ u_k, \text{ otherwise,} \end{cases} \tag{6.12}$$

where $\vec{p} = [p_1, \cdots, p_d]$ is the anchor point for $\vec{u}$. The above result can be rewritten as

$$\hat{g}_s(\vec{\xi}_s) = g\left(\tilde{\xi}\right), \text{ with } \tilde{\xi}_k = \begin{cases} \lambda_k(q_k), \text{ if } k \in \bar{s} \\ \lambda_k(\xi_k), \text{ otherwise,} \end{cases} \tag{6.13}$$

from which we can obtain $g_s(\vec{\xi}_s)$ defined in (6.6). Consequently, the decomposition for $g(\vec{\xi})$ can be obtained by using $\vec{q} = [\lambda_1(p_1), \cdots, \lambda_d(p_d)]$ as an anchor point of $\vec{\xi}$. □

Anchor point selection. It is is important to select a proper anchor point [148]. In circuit and MEMS applications, we find that $\vec{q} = \mathbb{E}(\vec{\xi})$ is a good choice.

Adaptive Implementation. In order to further reduce the computational cost, the truncated ANOVA decomposition (6.8) can be implemented in an adaptive way. Specifically, in practical computation we can ignore those terms that have small variance values. Such a treatment can produce a highly sparse generalized polynomial-chaos expansion.

For a given effective dimension $d_{\text{eff}} \ll d$, let

$$\mathcal{S}_k = \{s | s \subset \mathcal{I}, |s| = k\}, \; k = 1, \cdots d_{\text{eff}} \tag{6.14}$$

contain the initialized index sets for all k-variate terms in the ANOVA decomposition. Given an anchor point $\vec{q}$ and a threshold σ, starting from k=1, the main procedures of our ANOVA-based stochastic simulator are summarized below:

1. Compute g_0, which is a deterministic evaluation;

2. For every $s \in \mathcal{S}_k$, compute the low-dimensional function $g_s(\vec{\xi}_s)$ by stochastic

testing. The importance of $g_s(\vec{\xi}_s)$ is measured as

$$\theta_s = \frac{\mathbf{Var}\left(g_s\left(\vec{\xi}_s\right)\right)}{\sum_{j=1}^{k}\sum_{\tilde{s}\in S_j}\mathbf{Var}\left(g_{\tilde{s}}\left(\vec{\xi}_{\tilde{s}}\right)\right)}. \tag{6.15}$$

3. Update the index sets if $\theta_s < \sigma$ for $s \in \mathcal{S}_k$. Specifically, for $k < j \leq d_{\text{eff}}$, we check its index set $s' \in \mathcal{S}_j$. If s' contains all elements of s, then we remove s' from $\mathcal{S}_j$. Once s' is removed, we do not need to evaluate $g_{s'}(\vec{\xi}_{s'})$ in the subsequent computation.

4. Set $k = k+1$, and repeat steps 2) and 3) until $k = d_{\text{def}}$.

Example 4. *Let $y=g(\vec{\xi})$, $\vec{\xi} \in \mathbb{R}^{20}$ and $d_{\text{eff}} = 2$. Anchored ANOVA starts with*

$$\mathcal{S}_1 = \{\{j\}\}_{j=1,\cdots,20} \text{ and } \mathcal{S}_2 = \{\{j,k\}\}_{1\leq j<k\leq 20}.$$

For k=1, we first utilize stochastic testing to calculate $g_s(\vec{\xi}_s)$ and θ_s for every $s \in \mathcal{S}_1$. Assume

$$\theta_{\{1\}} > \sigma,\ \theta_{\{2\}} > \sigma, \text{ and } \theta_{\{j\}} < \sigma \text{ for all } j > 2,$$

implying that only the first two parameters are important to the output. Then, we only consider the coupling of ξ_1 and ξ_2 in $\mathcal{S}_2$, leading to

$$\mathcal{S}_2 = \{\{1,2\}\}.$$

Consequently, for $k = 2$ we only need to calculate one bi-variate function $g_{\{1,2\}}(\xi_1,\xi_2)$, yielding

$$\begin{aligned} g\left(\vec{\xi}\right) &\approx g_0 + \sum_{s\in S_1} g_s\left(\vec{\xi}_s\right) + \sum_{s\in S_2} g_s\left(\vec{\xi}_s\right) \\ &= g_0 + \sum_{j=1}^{20} g_{\{j\}}\left(\xi_j\right) + g_{\{1,2\}}\left(\xi_1,\xi_2\right). \end{aligned}$$

The pseudo codes of our implementation are summarized in Alg. 2. Lines 10 to 15 shows how to adaptively select the index sets. Let the final size of $\mathcal{S}_k$ be $|\mathcal{S}_k|$ and the total polynomial order in the stochastic testing simulator be p, then the total number

Algorithm 2 Stochastic Testing Circuit/MEMS Simulator Based on Adaptive Anchored ANOVA.

Initialize $\mathcal{S}_k$'s and set $\beta = 0$;
At the anchor point, run a deterministic circuit/MEMS simulation to obtain g_0, and set $y = g_0$;
for $k = 1, \cdots, d_{\text{eff}}$ **do**
  for each $s \in \mathcal{S}_k$ **do**
    run stochastic testing simulator to get the generalized polynomial-chaos expansion of $\hat{g}_s(\vec{\xi}_s)$;
    get the generalized polynomial-chaos expansion of $g_s(\vec{\xi}_s)$ according to (6.6);
    update $\beta = \beta + \mathbf{Var}\left(g_s(\vec{\xi}_s)\right)$;
    update $y = y + g_s(\vec{\xi}_s)$;
  end for
  for each $s \in \mathcal{S}_k$ **do**
    $\theta_s = \mathbf{Var}\left(g_s(\vec{\xi}_s)\right)/\beta$;
    if $\theta_s < \sigma$
      for any index set $s' \in \mathcal{S}_j$ with $j > k$, remove s' from $\mathcal{S}_j$ if $s \subset s'$.
    end if
  end for
end for

of samples used in Alg. 2 is

$$N = 1 + \sum_{k=1}^{d_{\text{eff}}} |\mathcal{S}_k| \frac{(k+p)!}{k!p!}. \tag{6.16}$$

Note that all univariate terms in ANOVA (i.e., $|s| = 1$) are kept in our implementation. For most circuit and MEMS problems, setting the effective dimension as 2 or 3 can achieve a high accuracy due to the weak couplings among different random parameters. For many cases, the univariate terms dominate the output of interest, leading to a near-linear complexity with respect to the parameter dimensionality d.

Remarks. Anchored ANOVA works very well for a large class of MEMS and circuit problems. However, in practice we also find a small number of examples (e.g., CMOS ring oscillators) that cannot be solved efficiently by the proposed algorithm, since many random variables affect significantly the output of interest. For such problems, it is possible to reduce the number of dominant random variables by a

linear transform [150] before applying anchored ANOVA. Other techniques such as compressed sensing can also be utilized to extract highly sparse surrogate models [88, 89, 151, 152] in the low-level simulation of our proposed hierarchical framework.

Global Sensitivity Analysis. Since each term $g_s(s_s)$ is computed by stochastic testing, Algorithm 2 provides a sparse generalized polynomial-chaos expansion for the output of interest: $y = \sum_{|\vec{\alpha}| \leq p} y_{\vec{\alpha}} H_{\vec{\alpha}}(\vec{\xi})$, where most coefficients are zero. From this result, we can identify how much each parameter contributes to the output by global sensitivity analysis. Two kinds of sensitivity information can be used to measure the importance of parameter ξ_k: the main sensitivity S_k and total sensitivity T_k, as computed below:

$$S_k = \frac{\sum_{\alpha_k \neq 0, \alpha_{j \neq k} = 0} |y_{\vec{\alpha}}|^2}{\mathbf{Var}(y)}, \quad T_k = \frac{\sum_{\alpha_k \neq 0} |y_{\vec{\alpha}}|^2}{\mathbf{Var}(y)}. \tag{6.17}$$

6.3 Enabling High-Level Simulation by Tensor-Train Decomposition

In this section, we show how to accelerate the high-level non-Monte-Carlo simulation by handling the obtained high-dimensional surrogate models with tensor-train decomposition [65–67].

6.3.1 Tensor-Based Three-Term Recurrence Relation

In order to obtain the orthonormal polynomials and Gauss quadrature points/weights of ζ, we must implement the three-term recurrence relation in (2.4). The main bottleneck is to compute the integrals in (6.3), since the probability density function of ζ is unknown.

For simplicity, we rewrite the integrals in (6.3) as $\mathbb{E}(q(\zeta))$, with $q(\zeta) = \phi_j^2(\zeta)$ or $q(\zeta) = \zeta \phi_j^2(\zeta)$. Since the probability density function of ζ is not given, we compute

the integral in the parameter space Ω:

$$\mathbb{E}\left(q\left(\zeta\right)\right) = \int_{\Omega} q\left(f\left(\vec{\xi}\right)\right)\rho(\vec{\xi})d\xi_1\cdots d\xi_d, \tag{6.18}$$

where $f(\vec{\xi})$ is a sparse generalized polynomial-chaos expansion for ζ obtained by

$$\zeta = f(\vec{\xi}) = \frac{(y-\mathbb{E}(y))}{\sqrt{\mathbf{Var}(y)}} = \sum_{|\vec{\alpha}|\leq p} \hat{y}_{\vec{\alpha}} H_{\vec{\alpha}}(\vec{\xi}). \tag{6.19}$$

We compute the integral in (6.18) with the following steps:

1. We utilize a multi-dimensional Gauss quadrature rule:

$$\mathbb{E}\left(q\left(\zeta\right)\right) \approx \sum_{i_1=1}^{m_1}\cdots\sum_{i_d=1}^{m_d} q\left(f\left(\xi_1^{i_1},\cdots,\xi_d^{i_d}\right)\right)\prod_{k=1}^{d} w_k^{i_k} \tag{6.20}$$

   where m_k is the number of quadrature points for ξ_k, $(\xi_k^{i_k}, w_k^{i_k})$ denotes the i_k-th Gauss quadrature point and weight.

2. We define two d-mode tensors $\boldsymbol{\mathcal{Q}}, \boldsymbol{\mathcal{W}} \in \mathbb{R}^{m_1\times m_2\cdots\times m_d}$, with each element defined as

$$\begin{aligned} \boldsymbol{\mathcal{Q}}\left(i_1,\cdots i_d\right) &= q\left(f\left(\xi_1^{i_1},\cdots,\xi_d^{i_d}\right)\right), \\ \boldsymbol{\mathcal{W}}\left(i_1,\cdots i_d\right) &= \prod_{k=1}^{d} w_k^{i_k}, \end{aligned} \tag{6.21}$$

   for $1 \leq i_k \leq m_k$. Now we can rewrite (6.20) as the inner product of $\boldsymbol{\mathcal{Q}}$ and $\boldsymbol{\mathcal{W}}$:

$$\mathbb{E}\left(q\left(\zeta\right)\right) \approx \left\langle \boldsymbol{\mathcal{Q}}, \boldsymbol{\mathcal{W}} \right\rangle. \tag{6.22}$$

   For simplicity, we set m_k=m in this manuscript.

The cost of computing the tensors and the tensor inner product is $O(m^d)$, which becomes intractable when d is large. Fortunately, both $\boldsymbol{\mathcal{Q}}$ and $\boldsymbol{\mathcal{W}}$ have low tensor ranks in our applications, and thus the high-dimensional integration (6.18) can be computed very efficiently in the following way:

1. **Low-rank representation of $\mathcal{W}$.** $\mathcal{W}$ can be written as a rank-1 tensor

$$\mathcal{W} = \mathbf{w}^{(1)} \circ \mathbf{w}^{(2)} \cdots \circ \mathbf{w}^{(d)}, \tag{6.23}$$

where $\mathbf{w}^{(k)} = [w_k^1; \cdots; w_k^m] \in \mathbb{R}^{m \times 1}$ contains all Gauss quadrature weights for parameter ξ_k. Clearly, now we only need $O(md)$ memory to store $\mathcal{W}$.

2. **Low-rank approximation for $\mathcal{Q}$.** $\mathcal{Q}$ can be well approximated by $\hat{\mathcal{Q}}$ with high accuracy in a tensor-train format [65–67]:

$$\hat{\mathcal{Q}}(i_1, \cdots i_d) = \mathcal{G}_1(:, i_1, :)\, \mathcal{G}_2(:, i_1, :) \cdots \mathcal{G}_d(:, i_d, :) \tag{6.24}$$

with a pre-selected error bound ϵ such that

$$\left\| \mathcal{Q} - \hat{\mathcal{Q}} \right\|_F \leq \varepsilon \left\| \mathcal{Q} \right\|_F. \tag{6.25}$$

For many circuit and MEMS problems, a tensor train with very small TT-ranks can be obtained even when $\epsilon = 10^{-12}$ (which is very close to the machine precision).

3. **Fast computation of (6.22).** With the above low-rank tensor representations, the inner product in (6.22) can be accurately estimated as

$$\left\langle \hat{\mathcal{Q}}, \mathcal{W} \right\rangle = \mathbf{T}_1 \cdots \mathbf{T}_d, \text{ with } \mathbf{T}_k = \sum_{i_k=1}^{m} w_k^{i_k} \mathcal{G}_k(:, i_k, :) \tag{6.26}$$

Now the cost of computing the involved high-dimensional integration dramatically reduces to $O(dmr^2)$, which only linearly depends the parameter dimensionality d.

6.3.2 Efficient Tensor-Train Computation

Now we discuss how to obtain a low-rank tensor train. An efficient implementation called **TT_cross** is described in [67] and included in the public-domain MATALB

package **TT_Toolbox** [153]. In **TT_cross**, Skeleton decomposition is utilized to compress the TT-rank r_k by iteratively searching a rank-r_k maximum-volume submatrix when computing $\mathcal{G}_k$. A major advantage of **TT_cross** is that we do not need to know $\mathcal{Q}$ *a-priori*. Instead, we only need to specify how to evaluate the element $\mathcal{Q}(i_1, \cdots, i_d)$ for a given index $(i_1, \cdots, i_d)$. As shown in [67], with Skeleton decompositions a tensor-train decomposition needs $O(ldmr^2)$ element evaluations, where l is the number of iterations in a Skeleton decomposition. For example, when $l = 10$, $d = 50$, $m = 10$ and $r = 4$ we may need up to 10^5 element evaluations, which can take about one hour since each element of $\mathcal{Q}$ is a high-order polynomial function of many bottom-level random variables $\vec{\xi}$.

In order to make the tensor-train decomposition of $\mathcal{Q}$ fast, we employ some tricks to evaluate more efficiently each element of $\mathcal{Q}$. The details are given below.

- **Fast evaluation of $\mathcal{Q}(i_1, \cdots, i_d)$.** In order to reduce the cost of evaluating $\mathcal{Q}(i_1, \cdots, i_d)$, we first construct a low-rank tensor train $\hat{\mathcal{A}}$ for the intermediate-level random parameter ζ, such that

$$\left\| \mathcal{A} - \hat{\mathcal{A}} \right\|_F \leq \varepsilon \left\| \mathcal{A} \right\|_F, \ \mathcal{A}(i_1, \cdots, i_d) = f\left(\xi_1^{i_1}, \cdots, \xi_d^{i_d}\right).$$

  Once $\hat{\mathcal{A}}$ is obtained, $\mathcal{Q}(i_1, \cdots, i_d)$ can be evaluated by

$$\mathcal{Q}(i_1, \cdots, i_d) \approx q\left(\hat{\mathcal{A}}(i_1, \cdots, i_d)\right), \tag{6.27}$$

  which reduces to a cheap low-order univariate polynomial evaluation. However, computing $\hat{\mathcal{A}}(i_1, \cdots, i_d)$ by directly evaluating $\mathcal{A}(i_1, \cdots, i_d)$ in **TT_cross** can be time-consuming, since $\zeta = f(\vec{\xi})$ involves many multivariate basis functions.

- **Fast evaluation of $\mathcal{A}(i_1, \cdots, i_d)$.** The evaluation of $\mathcal{A}(i_1, \cdots, i_d)$ can also be accelerated by exploiting the special structure of $f(\vec{\xi})$. It is known that the generalized polynomial-chaos basis of $\vec{\xi}$ is

$$H_{\vec{\alpha}}\left(\vec{\xi}\right) = \prod_{k=1}^{d} \varphi_{\alpha_k}^{(k)}(\xi_k), \ \vec{\alpha} = [\alpha_1, \cdots, \alpha_d] \tag{6.28}$$

where $\varphi_{\alpha_k}^{(k)}(\xi_k)$ is the degree-α_k orthonormal polynomial of ξ_k, with $0 \leq \alpha_k \leq p$. We first construct a 3-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{d\times(p+1)\times m}$ indexed by (k, α_k+1, i_k) with

$$\boldsymbol{\mathcal{X}}(k, \alpha_k+1, i_k) = \varphi_{\alpha_k}^{(k)}\left(\xi_k^{i_k}\right) \tag{6.29}$$

where $\xi_k^{i_k}$ is the i_k-th Gauss quadrature point for parameter ξ_k [as also used in (6.20)]. Then, each element of $\boldsymbol{\mathcal{A}}(i_1, \cdots, i_d)$ can be calculated efficiently as

$$\boldsymbol{\mathcal{A}}(i_1, \cdots, i_d) = \sum_{|\vec{\alpha}|<p} \vec{y}_{\vec{\alpha}} \prod_{k=1}^{d} \boldsymbol{\mathcal{X}}(k, \alpha_k+1, i_k) \tag{6.30}$$

without evaluating the multivariate polynomials. Constructing $\boldsymbol{\mathcal{X}}$ does not necessarily need $d(p+1)m$ polynomial evaluations, since the matrix $\boldsymbol{\mathcal{X}}(k,:,:)$ can be reused for any other parameter ξ_j that has the same type of distribution with ξ_k.

In summary, we compute a tensor-train decomposition for $\boldsymbol{\mathcal{Q}}$ as follows: 1) we construct the 3-mode tensor $\boldsymbol{\mathcal{X}}$ defined in (6.29); 2) we call **TT_cross** to compute $\hat{\boldsymbol{\mathcal{A}}}$ as a tensor-train decomposition of $\boldsymbol{\mathcal{A}}$, where (6.30) is used for fast element evaluation; 3) we call **TT_cross** again to compute $\hat{\boldsymbol{\mathcal{Q}}}$, where (6.27) is used for the fast element evaluation of $\boldsymbol{\mathcal{Q}}$. With the above fast tensor element evaluations, the computation time of **TT_cross** can be reduced from dozens of minutes to several seconds to generate some accurate low-rank tensor trains for our high-dimensional surrogate models.

6.3.3 Algorithm Summary

Given the Gauss quadrature rule for each bottom-level random parameter ξ_k, our tensor-based three-term recurrence relation for an intermediate-level random parameter ζ is summarized in Alg. 3. This procedure can be repeated for all ζ_i's to obtain their univariate generalized polynomial-chaos basis functions and Gauss quadrature rules, and then the stochastic testing simulator [34–36] (and any other standard stochastic spectral method [40,54,93]) can be employed to perform high-level stochas-

Algorithm 3 Tensor-based generalized polynomial-chaos basis and Gauss quadrature rule construction for ζ.

1: Initialize: $\phi_0(\zeta) = \pi_0(\zeta) = 1$, $\phi_1(\zeta) = \pi_1(\zeta) = \zeta$, $\kappa_0 = \kappa_1 = 1$, $\gamma_0 = 0$, $a = 1$;
2: Compute a low-rank tensor train $\hat{\boldsymbol{\mathcal{A}}}$ for ζ;
3: Compute a low-rank tensor train $\hat{\boldsymbol{\mathcal{Q}}}$ for $q(\zeta) = \zeta^3$, and obtain $\gamma_1 = \left\langle \hat{\boldsymbol{\mathcal{Q}}}, \boldsymbol{\mathcal{W}} \right\rangle$ via (6.26);
4: **for** $j = 2, \cdots, p$ **do**
5: get $\pi_j(\zeta) = (\zeta - \gamma_{j-1})\pi_{j-1}(\zeta) - \kappa_{j-1}\pi_{j-2}(\zeta)$;
6: construct a low-rank tensor train $\hat{\boldsymbol{\mathcal{Q}}}$ for $q(\zeta) = \pi_j^2(\zeta)$, and compute $\hat{a} = \left\langle \hat{\boldsymbol{\mathcal{Q}}}, \boldsymbol{\mathcal{W}} \right\rangle$ via (6.26) ;
7: $\kappa_j = \hat{a}/a$, and update $a = \hat{a}$;
8: construct a low-rank tensor train $\hat{\boldsymbol{\mathcal{Q}}}$ for $q(\zeta) = \zeta\pi_j^2(\zeta)$, and compute $\gamma_j = \left\langle \hat{\boldsymbol{\mathcal{Q}}}, \boldsymbol{\mathcal{W}} \right\rangle / a$;
9: normalization: $\phi_j(\zeta) = \frac{\pi_j(\zeta)}{\sqrt{\kappa_0 \cdots \kappa_j}}$;
10: **end for**
11: Form matrix $\mathbf{J}$ in (2.12);
12: Eigenvalue decomposition: $\mathbf{J} = \mathbf{U}\Sigma\mathbf{U}^T$;
13: Compute the Gauss-quadrature abscissa $\zeta^j = \Sigma(j,j)$ and weight $w^j = (\mathbf{U}(1,j))^2$ for $j = 1, \cdots, p+1$;

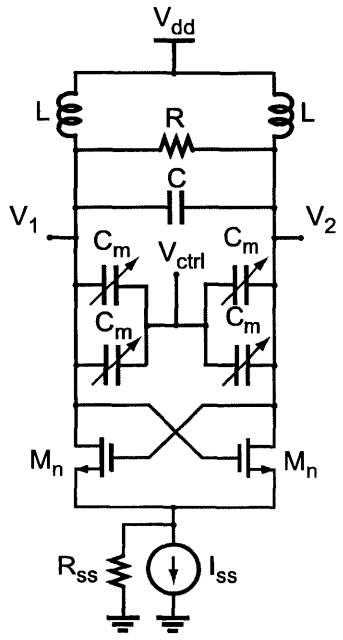
tic simulation.

Remarks. 1) If the outputs of a group of subsystems are identically independent, we only need to run Alg. 3 once and reuse the results for the other subsystems in the group. 2) When there exist many subsystems, our ANOVA-based stochastic solver may also be utilized to accelerate the high-level simulation.

6.4 Numerical Results of a High-Dimensional MEMS/IC Co-Design

6.4.1 MEMS/IC Example

In order to demonstrate the application of our hierarchical uncertainty quantification in high-dimensional problems, we consider the oscillator circuit shown in Fig. 6-7. This oscillator has four identical RF MEMS switches acting as tunable capacitors. The MEMS device used in this paper is a prototyping model of the RF MEMS



Parameter	Value
V_{dd}	2.5V
V_{ctrl}	0-2.5V
C	10.83fF
R	110Ω
L	8.8nH
$(W/L)_n$	4/0.25
I_{ss}	0.5mA
R_{ss}	$10^6\Omega$

Figure 6-7: Schematic of the oscillator circuit with 4 MEMS capacitors (denoted as C_m), with 184 random parameters in total.

Table 6.1: Different hierarchical simulation methods.

Method	Low-level simulation	High-level simulation
Proposed	Alg. 2	stochastic testing [36]
Method 1 [1]	Monte Carlo	Monte Carlo
Method 2	Alg. 2	Monte Carlo

capacitor reported in [154, 155].

Since the MEMS switch has a symmetric structure, we construct a model for only half of the design, as shown in Fig. 6-8. The simulation and measurement results in [33] show that the pull-in voltage of this MEMS switch is about 37 V. When the control voltage is far below the pull-in voltage, the MEMS capacitance is small and almost constant. In this paper, we set the control voltage to 2.5 V, and thus the MEMS switch can be regarded as a small linear capacitor. As already shown in [31], the performance of this MEMS switch can be influenced significantly by process variations.

In our numerical experiments, we use 46 independent random parameters with Gaussian and Gamma distributions to describe the material (e.g, conductivity and

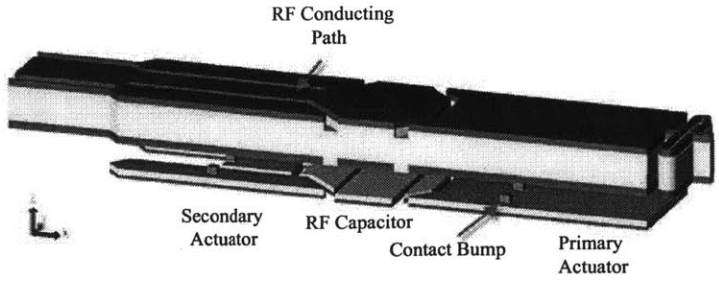


Figure 6-8: 3-D schematic of the RF MEMS capacitor.

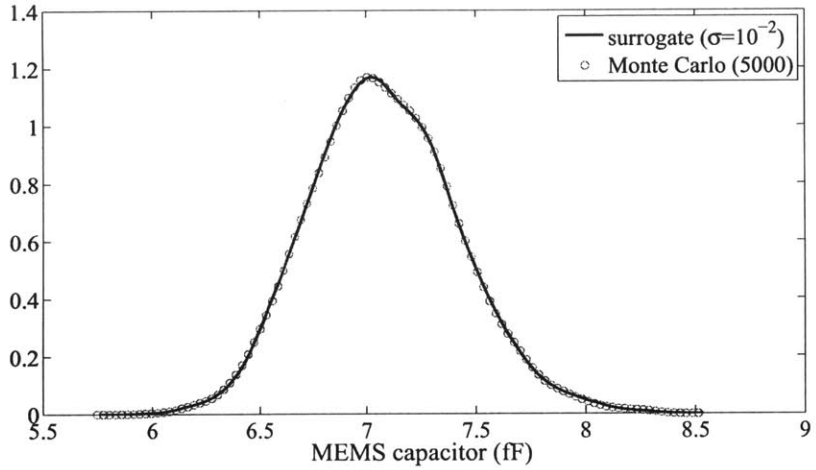


Figure 6-9: Comparison of the density functions obtained by our surrogate model and by 5000-sample Monte Carlo analysis of the original MEMS equation.

dielectric constants), geometric (e.g., thickness of each layer, width and length of each mechanical component) and environmental (e.g., temperature) uncertainties of each switch. For each random parameter, we assume that its standard deviation is 3% of its mean value. In the whole circuit, we have 184 random parameters in total. Due to such high dimensionality, simulating this circuit by stochastic spectral methods is a challenging task.

In the following experiments, we simulate this challenging design case using our proposed hierarchical stochastic spectral methods. We also compare our algorithm with other two kinds of hierarchical approaches listed in Table 6.1. In Method 1, both

Table 6.2: Surrogate model extraction with different σ values.

σ	# $\|s\|=1$	# $\|s\|=2$	# $\|s\|=3$	# ANOVA	# nonzero gPC	# samples
0.5	46	0	0	47	81	185
0.1 to 10^{-3}	46	3	0	50	90	215
10^{-4}	46	10	1	58	112	305
10^{-5}	46	21	1	69	144	415

low-level and high-level simulations use Monte Carlo, as suggested by [1]. In Method 2, the low-level simulation uses our ANOVA-based sparse simulator (Alg. 2), and the high-level simulation uses Monte Carlo.

6.4.2 Surrogate Model Extraction

In order to extract an accurate surrogate model for the MEMS capacitor, Alg. 2 is implemented in the commercial network-based MEMS simulation tool MEMS+ [143] of Coventor Inc. Each MEMS switch is described by a stochastic differential equation [c.f. (1.3)] with consideration of process variations. In order to compute the MEMS capacitor, we can ignore the derivative terms and solve for the static solutions.

By setting $\sigma = 10^{-2}$, our ANOVA-based stochastic MEMS simulator generates a sparse 3rd-order generalized polynomial chaos expansion with only 90 non-zero coefficients, requiring only 215 simulation samples and 8.5 minutes of CPU time in total. This result has only 3 bivariate terms and no three-variable terms in ANOVA decomposition, due to the very weak couplings among different random parameters. Setting $\sigma = 10^{-2}$ can provide a highly accurate generalized polynomial chaos expansion for the MEMS capacitor, which has a relative error around 10^{-6} (in the L_2 sense) compared to that obtained by setting $\sigma = 10^{-5}$.

By evaluating the surrogate model and the original model (by simulating the original MEMS equation) with 5000 samples, we have obtained the same probability density curves shown in Fig. 6-9. Note that using the standard stochastic testing simulator [34–36] requires 18424 basis functions and simulation samples for this high-dimensional example, which is prohibitively expensive on a regular computer. When the effective dimension d_{eff} is set as 3, there should be 16262 terms in the truncated ANOVA decomposition (6.8). However, due to the weak couplings among different

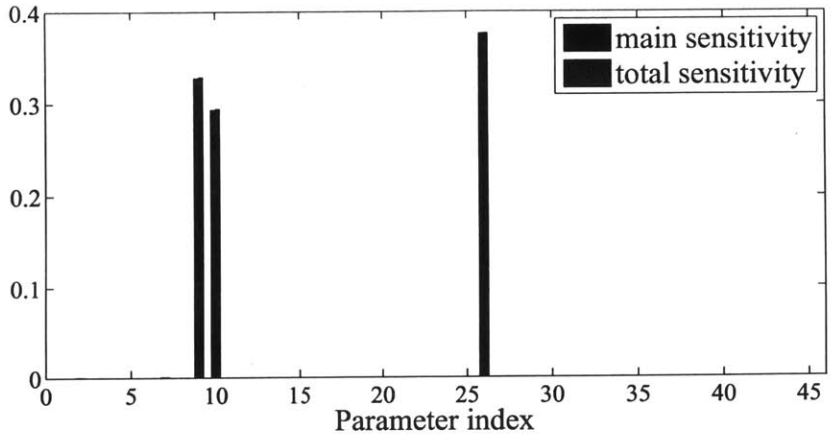


Figure 6-10: Main and total sensitivities of different random parameters for the RF MEMS capacitor.

random parameters, only 90 of them are non-zero.

We can get surrogate models with different accuracies by changing the threshold σ. Table 6.2 has listed the number of obtained ANOVA terms, the number of non-zero generalized polynomial chaos (gPC) terms and the number of required simulation samples for different values of σ. From this table, we have the following observations:

1. When σ is large, only 46 univariate terms (i.e., the terms with $|s| = 1$) are obtained. This is because the variance of all univariate terms are regarded as small, and thus all multivariate terms are ignored.

2. When σ is reduced (for example, to 0.1), three dominant bivariate terms (with $|s| = 2$) are included by considering the coupling effects of the three most influential random parameters. Since the contributions of other parameters are insignificant, the result does not change even if σ is further decreased to 10^{-3}.

3. A three-variable term (with $|s| = 3$) and some bivariate coupling terms among other parameters can only be captured when σ is reduced to 10^{-4} or below. In this case, the effect of some non-dominant parameters can be captured.

Fig. 6-10 shows the global sensitivity of this MEMS capacitor with respect to all 46 random parameters. The output is dominated by only 3 parameters. The

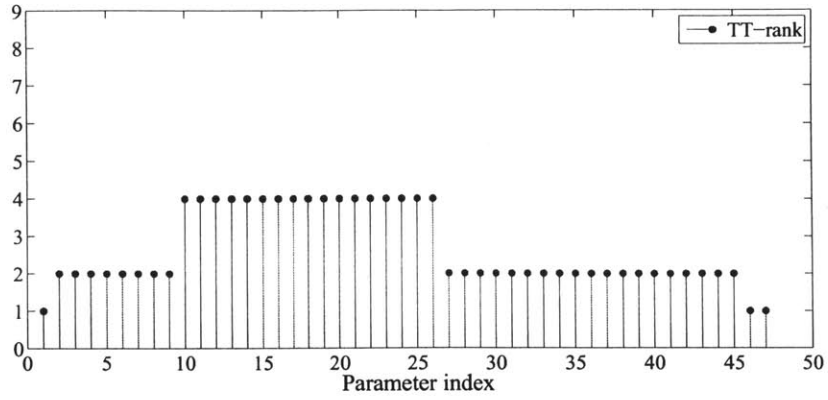


Figure 6-11: TT-rank for the surrogate model of the RF MEMS capacitor.

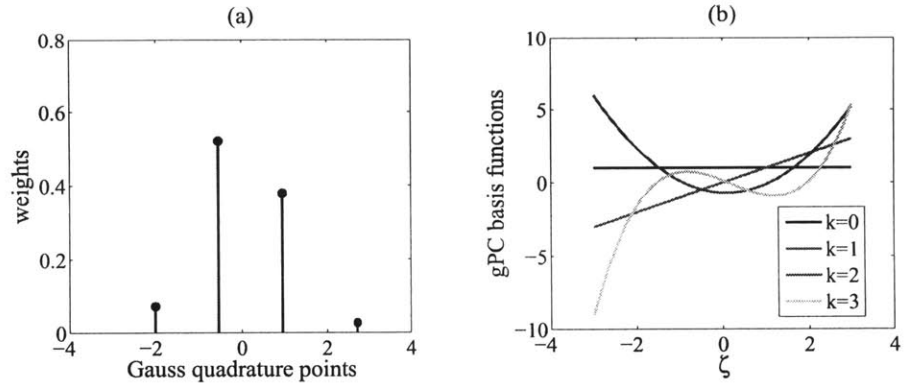


Figure 6-12: (a) Gauss quadrature rule and (b) generalized polynomial chaos (gPC) basis functions for the RF MEMS capacitor.

other 43 parameters contribute to only 2% of the capacitor's variance, and thus their main and total sensitivities are almost invisible in Fig. 6-10. This explains why the generalized polynomial-chaos expansion is highly sparse. Similar results have already been observed in the statistical analysis of CMOS analog circuits [37].

6.4.3 High-Level Simulation

The surrogate model obtained with $\sigma = 10^{-2}$ is imported into the stochastic testing circuit simulator described in [34–36] for high-level simulation. At the high-level, we

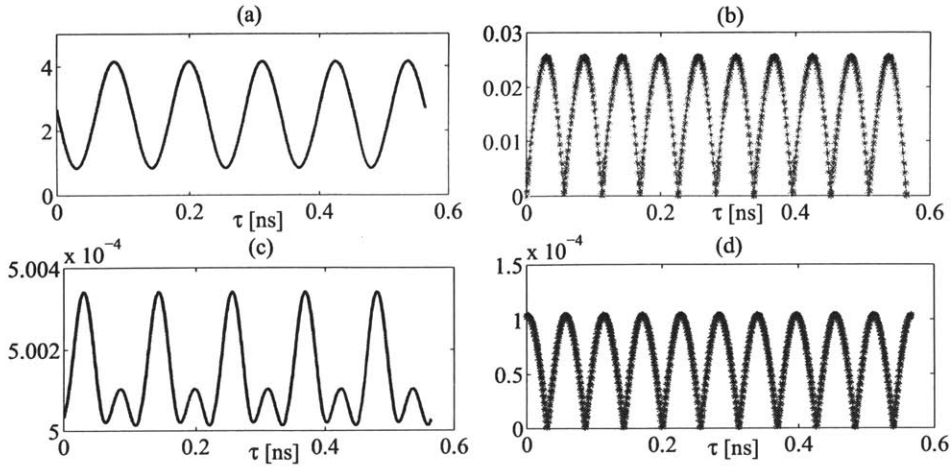


Figure 6-13: Simulated waveforms on the scaled time axis $\tau = t/a(\vec{\zeta})$. (a) and (b): the mean and standard deviation of V_{out1} (unit: V), respectively; (c) and (d): the mean and standard deviation of the current (unit: A) from V_{dd}, respectively.

have a stochastic DAE to describe the oscillator

$$\frac{d\vec{q}\left(\vec{x}(t,\vec{\zeta}),\vec{\xi}\right)}{dt} + \vec{f}\left(\vec{x}(t,\vec{\zeta}),\vec{\zeta},u\right) = 0 \tag{6.31}$$

where the input signal u is constant, $\vec{\zeta}=[\zeta_1,\cdots,\zeta_4] \in \mathbb{R}^4$ are the intermediate-level random parameters describing the four MEMS capacitors. Since the oscillation period $T(\vec{\zeta})$ now depends on the MEMS capacitors, the periodic steady-state can be written as $\vec{x}(t,\zeta) = \vec{x}(t+T(\vec{\zeta}),\zeta)$. We simulate the stochastic oscillator by the algorithm in Chapter 3. The scaled waveform $z(\tau,\vec{\zeta})$ is computed and then mapped onto the original time axis t.

In order to apply stochastic testing at the high level, we need to compute some specialized orthonormal polynomials and Gauss quadrature points for each intermediate-level parameter ζ_i. We use 9 quadrature points for each bottom-level parameter ξ_k to evaluate the high-dimensional integrals involved in the three-term recurrence relation. This leads to 9^{46} function evaluations at all quadrature points, which is prohibitively expensive.

In order to handle the high-dimensional MEMS surrogate models, the following

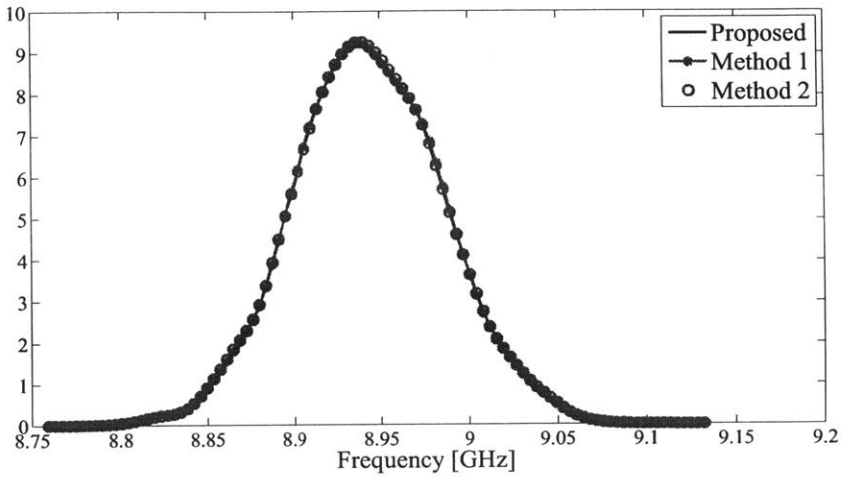


Figure 6-14: Probability density functions of the oscillation frequency.

tensor-based procedures are employed:

- With Alg. 3, a low-rank tensor train of ζ_1 is first constructed for an MEMS capacitor. For most dimensions the rank is only 2, and the highest rank is 4, as shown in Fig. 6-11.

- Using the obtained tensor train, the Gauss quadrature points and generalized polynomial chaos basis functions are efficiently computed, as plotted in Fig. 6-12.

The total CPU time for constructing the tensor trains and computing the basis functions and Gauss quadrature points/weights is about 40 seconds in MATALB. If we directly evaluate the high-dimensional multivariate generalized polynomial-chaos expansion, the three-term recurrence relation requires almost 1 hour. The obtained results can be reused for all MEMS capacitors since they are independently identical.

With the obtained basis functions and Gauss quadrature points/weights for each MEMS capacitor, the stochastic periodic steady-state solver [36] is called at the high level to simulate the oscillator. Since there are 4 intermediate-level parameters ζ_i's, only 35 basis functions and testing samples are required for a 3rd-order generalized

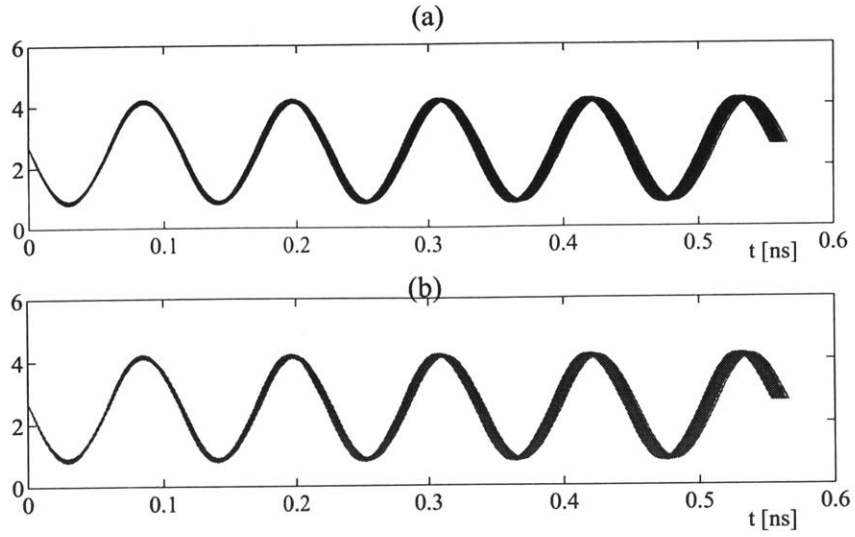


Figure 6-15: Realization of the output voltages (unit: volt) at 100 bottom-level samples, generated by (a) proposed method and (b) Method 1.

polynomial-chaos expansion, leading to a simulation cost of only 56 seconds in MATLAB.

Fig. 6-13 shows the waveforms from our algorithm at the scaled time axis $\tau = t/a(\vec{\zeta})$. The high-level simulation generates a generalized polynomial-chaos expansion for all nodal voltages, branch currents and the exact parameter-dependent period. Evaluating the resulting generalized polynomial-chaos expansion with 5000 samples, we have obtained the density function of the frequency, which is consistent with those from Method 1 (using 5000 Monte Carlo samples at both levels) and Method 2 (using Alg. 1 at the low level and using 5000 Monte-Carlo samples at the high level), as shown in Fig. 6-14.

In order to show the variations of the waveform, we further plot the output voltages for 100 bottom-level random samples. As shown in Fig. 6-15, the results from our proposed method and from Method 1 are indistinguishable from each other.

Table 6.3: CPU times of different hierarchical stochastic simulation algorithms.

Simulation Method	Low level		High level		Total simulation cost
	Method	CPU time	Method	CPU time	
Proposed	Alg. 2	8.5 min	stochastic testing	1.5 minute	Low (10 min)
Method 1	Monte Carlo	13.2 h	Monte Carlo	2.2 h	High (15.4 h)
Method 2	Alg. 2	8.5 min	Monte Carlo	2.2 h	Medium (2.3 h)

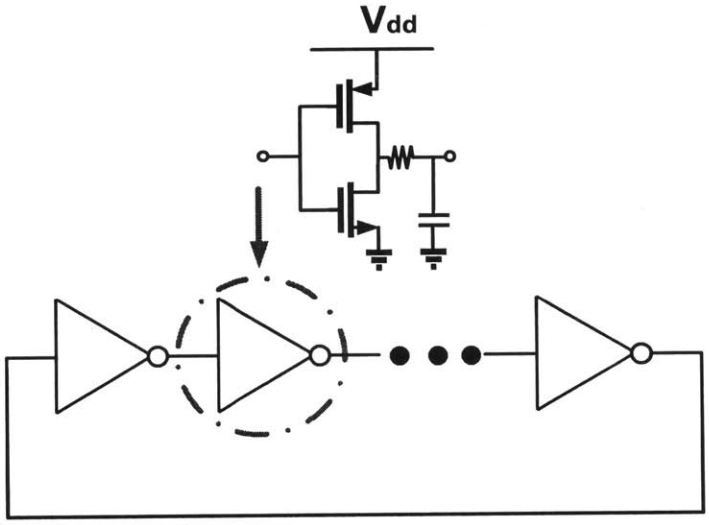


Figure 6-16: Schematic of a 7-stage CMOS ring oscillator.

6.4.4 Computational Cost

Table 6.3 has summarized the performances of all three methods. In all Monte Carlo analysis, 5000 random samples are utilized. If Method 1 [1] is used, Monte Carlo has to be repeatedly used for each MEMS capacitor, leading to extremely long CPU time due to the slow convergence. If Method 2 is used, the efficiency of the low-level surrogate model extraction can be improved due to the employment of generalized polynomial-chaos expansion, but the high-level simulation is still time-consuming. Since our proposed technique utilizes fast stochastic testing algorithms at both levels, this high-dimensional example can be simulated at very low computational cost, leading to 92× speedup over Method 1 and 14× speedup over Method 2.

6.5 Limitations and Possible Solutions

6.5.1 Limitation of Alg. 2

The ANOVA-based algorithm may become inefficient for a circuit with lots of important random parameters. For such a case, lots of multi-variable functions may have to be evaluated in Alg. 2. A typical example is CMOS ring oscillator, where each n-type (or p-type) transistor contributes equally to the frequency, and thus few random variables can be ignored. Consider the 7-stage ring oscillator shown in Fig. 6-16. Assume that for each transistor we have important four variations: threshold voltage, gate oxide thickness, effective width and length, resulting in 56 random parameters in total. We may find that none of these random parameters can be ignored after obtaining the uni-variate terms. As a result, 1520 bi-variate functions have to be computed even if we set the effective dimension as low as 2. Consequently, to obtain a 3rd-order generalized polynomial-chaos expansion, 15624 function values must be evaluated, which can be prohibitively expensive.

We suggest two possible solutions to this problem. First, one may rotate the parameter space such that in the rotated space only a few random parameters are important. Second, compressed sensing or machine learning techniques can be useful to obtain a sparse model even if rotating the parameter space is difficult or impossible.

6.5.2 Limitation of Alg. 3

Alg. 3 is efficient if the outputs of all subsystems have low tensor ranks. This may not be true for some cases. For example, when simulating the uncertainties of a phase-lock loop (c.f. Fig. 6-17), one needs to use both the frequency and frequency gain of the voltage-controlled oscillator (VCO) as the inputs of a system-level description. Using our stochastic simulator, a sparse and low-rank approximation for the oscillator's period can be obtained, but the corresponding frequency and frequency gain are not guaranteed to have low tensor ranks.

In order to make the high-level simulation efficient for high-dimensional cases,

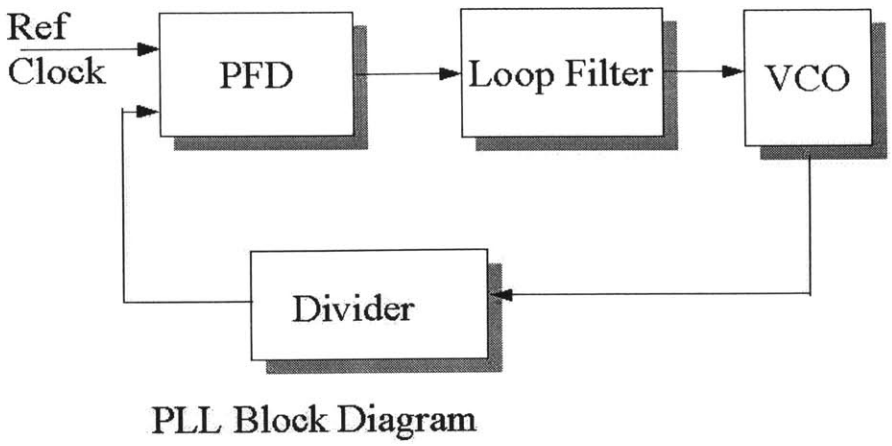


Figure 6-17: The block diagram of a phase-lock loop.

it is desirable to develop novel simulators that can guarantee sparse and low-rank properties simultaneously.

Chapter 7

Enabling Hierarchical Uncertainty Quantification by Density Estimation

In this chapter we develop an alternative approach to enable hierarchical uncertainty quantification. Instead of using fast multi-dimensional integration, this approach first computes the density function of each subsystem, and then computes the basis functions and Gauss quadrature rules required for high-level uncertainty quantification in an analytical way. Specifically, using two monotone interpolation schemes [156–159], physically consistent closed-form cumulative density functions and probability density functions are constructed for the output of each subsystem. Due to the special forms of the obtained density functions, we can determine a proper Gauss quadrature rule and the basis functions that further allow a generalized polynomial chaos expansion in system-level simulation.

Although more accuracy may be lost compared with the approach in Chapter 6, this alternative is useful even if the output of a subsystem has non-smooth dependence on process variations (for instance, when the subsystem is described by a parameterized or stochastic reduced-order model [18, 19, 141]). The density estimation suggested in this chapter also shows some better numerical properties over existing moment-matching techniques such as asymptotic probability extraction [2, 3].

7.1 Algorithms via Density Estimation

Assume that we have a general (and possibly non-smooth) surrogate model

$$\hat{x} = f(\vec{\xi}), \text{ with } \vec{\xi} \in \mathbb{R}^d \tag{7.1}$$

to represent the output of a subsystem in a complex system design, where $\hat{x}$ denote multiple mutually independent lower-level random parameters. We aim to approximate the density function of $\hat{x}$ such that a set of orthonormal polynomials and Gauss quadrature points/weights can be computed for high-level uncertainty quantification. The approximated density function should be physically consistent. In other words,

- The approximated probability density function should be non-negative;
- The obtained cumulative density function should be monotonic increasing from 0 to 1.

Both both kernel density estimation [160–162] and asymptotic probability extraction [2, 3] can be used to approximate a density function. Kernel density estimation is seldom used in circuit modeling due to several shortcomings. First, the approximated probability density function is not compact: one has to store all samples as the parameters of a density function, which is inefficient for reuse in a stochastic simulator. Second, it is not straightforward to generate samples from the approximated probability density function. Third, the accuracy of kernel density estimation highly depends on the specific forms of the kernel functions (although Gaussian kernel seems suitable for the examples used in this work) as well as some parameters (e.g., the smoothing parameter). In contrast, asymptotic probability extraction [2, 3] and its variant can efficiently approximate the density of $\hat{x}$ by moment matching, but it is numerically unstable and the obtained density function may be physically inconsistent [163]. Furthermore, asymptotic probability extraction has a strict restriction on the form of $f(\vec{\xi})$: $\vec{\xi}$ should be Gaussian and $f(\vec{\xi})$ should be very smooth (for instance, being a linear quadratic function).

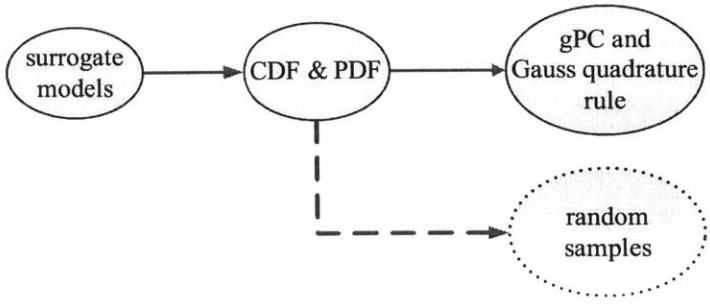


Figure 7-1: Construct generalized polynomial-chaos (gPC) bases and Gauss quadrature rules from surrogate models. Here CDF and PDF means "cumulative density function" and "probability density function", respectively.

We first employ the linear transformation

$$x = \frac{\hat{x} - a}{b} \tag{7.2}$$

to define a new random input x, which aims to improve the numerical stability. Once we obtain the cumulative density function and probability density function of x (denoted as $p(x)$ and $\rho(x)$, respectively), then the cumulative density function and probability density function of $\hat{x}$ can be obtained by

$$\hat{p}(\hat{x}) = p\left(\frac{\hat{x} - a}{b}\right) \text{ and } \hat{\rho}(\hat{x}) = \frac{1}{b}\rho(\frac{\hat{x} - a}{b}) \tag{7.3}$$

respectively.

As shown in Fig. 7-1, we first construct the density functions of x *in a proper way*, then we determine the generalized polynomial-chaos bases of x and a proper Gauss quadrature rule based on the obtained density functions. With the obtained cumulative density function, random samples of x could be easily obtained for higher-level Monte Carlo-based simulation, however such task is not the focus of this paper. Our proposed framework consists of the following steps.

- **Step 1.** Use N Monte Carlo samples (or readily available measurement/simulation

data) to obtain the discrete cumulative density function curve of $\hat{x} = f(\vec{\xi})$. Since $f(\vec{\xi})$ is a surrogate model, this step can be extremely efficient.

- **Step 2.** Let $\delta > 0$ be a small threshold value, $\hat{x}_{\min}$ and $\hat{x}_{\max}$ be the minimum and maximum values of $\hat{x}$ from the Monte Carlo analysis (or available data), respectively. We set $a{=}\hat{x}_{\min} - \delta$, $b{=}\hat{x}_{\max} + \delta - a$, then N samples of x in the interval $(0, 1)$ are obtained by the linear transformation (7.2). The obtained samples provide a discrete cumulative density function for x.

- **Step 3.** From the obtained cumulative density function curve of x, pick $n \ll N$ points (x_i, y_i) for $i = 1, \cdots, n$. Here x_i denotes the value of x, and y_i the corresponding cumulative density function value. The data are monotone: $x_i < x_{i+1}$ and $0 = y_1 \leq \cdots \leq y_n = 1$.

- **Step 4.** Use a monotone interpolation algorithm in Section 7.2 to construct a closed-form function $p(x)$ to approximate the cumulative density function of x.

- **Step 5.** Compute the first-order derivative of $p(x)$ and use it as a closed-form approximation to $\rho(x)$.

- **Step 6.** With the obtained $\rho(x)$, we utilize the procedures in Section 7.3 to construct the generalized polynomial-chaos basis functions and Gauss quadrature points/weights for x.

Many surrogate models are described by truncated generalized polynomial chaos expansions. The cost of evaluating such models may increase dramatically when the lower-level parameters $\vec{\xi}$ have a high dimensionality (which may occasionally happen), although the surrogate model evaluation is still much faster than the detailed simulation. Fortunately, in practical high-dimensional stochastic problems, normally only a small number of parameters are important to the output and most cross terms will vanish [88, 89, 164]. Consequently, a highly sparse generalized polynomial chaos expansion can be utilized for fast evaluation. Furthermore, when the coupling between the random parameters are weak, quasi-Monte Carlo [165] can further speed up the surrogate model evaluation.

In Step 3, we first select $(x_1, y_1) = (0, 0)$ and $(x_n, y_n) = (1, 1)$. The n data points are selected such that

$$|x_{i+1} - x_i| \leq \frac{1}{m} \text{ and } |y_{i+1} - y_i| \leq \frac{1}{m}, \tag{7.4}$$

where m is an integer used to control n. This constraint ensures that the interpolation points are selected properly such that the behavior around the peak of $\rho(x)$ is well captured. In practical implementation, for $k = 2, \cdots, n-1$, the point (x_k, y_k) is selected from the cumulative density function curve subject to the following criteria:

$$\sqrt{(y_{k-1} - y_k)^2 + (x_{k-1} - x_k)^2} \approx \frac{1}{m}. \tag{7.5}$$

For $x \notin [x_1, x_n]$, we set $\rho(x)$=0. This treatment introduces some errors in the tail regions. Approximating the tail regions is non-trivial, but such errors may be ignored if rare failure events are not a major concern (e.g., in the yield analysis of some analog/RF circuits).

Remark 3.1: Similar to standard stochastic spectral simulators [4, 34–36, 40, 41, 54–57, 93, 109, 166], this paper assumes that $\hat{x}_i$'s are mutually independent. It is more difficult to handle correlated and non-Gaussian random inputs. Not only is it difficult to construct the density functions, but also it is hard to construct the basis functions even if the multivariate density function is given [42, 47]. How to handle correlated non-Gaussian random inputs remains an open and important topic in uncertainty quantification [42]. Some of our progress in this direction will be reported in [167].

The most important parts of our algorithm are Step 4 and Step 6. In Section 7.2 we will show how we guarantee that the obtained density functions are physically consistent. Step 6 will be detailed in Section 7.3, with emphasis on an efficient analytical implementation.

7.2 Implementation of the Density Estimator

This section presents the numerical implementation of our proposed density estimation. Our implementation is based on two monotone interpolation techniques, which are well studied in the mathematical community but have not been applied to uncertainty quantification. Since we approximate the cumulative density function $p(x)$ in the interval $x \in [x_1, x_n]$, in both methods we set $p(x) = y_1 = 0$ for $x < x_1$ and $p(x) = y_n = 1$ for $x > x_n$, respectively.

7.2.1 Method 1: Piecewise Cubic Interpolation

Our first implementation uses a piecewise cubic interpolation [156, 157]. With the monotone data from Step 3 of Section 7.1, we construct $p(x)$ as a cubic polynomial:

$$p(x) = c_k^1 + c_k^2(x - x_k) + c_k^3(x - x_k)^2 + c_k^4(x - x_k)^3 \tag{7.6}$$

for $x \in [x_k, x_{k+1}]$, $0 < k < n$. If $y_k{=}y_{k+1}$, we simply set $c_k^1{=}y_k$ and $c_k^2{=}c_k^3{=}c_k^4{=}\ 0$. Otherwise, the coefficients are selected according to the following formula [157]

$$c_k^1 = y_k,\ \ c_k^2 = \dot{y}_k,\ c_k^3 = \frac{s_k - \dot{y}_{k+1} - 2\dot{y}_k}{\Delta x_k},\ c_k^4 = \frac{2s_k - \dot{y}_{k+1} - \dot{y}_k}{(\Delta x_k)^2} \tag{7.7}$$

where $\Delta x_k{=}x_{k+1}{-}x_k$, $s_k{=}\frac{y_{k+1}-y_k}{\Delta x_k}$. This formula ensures that $p(x)$ and $p'(x)$ are continuous, $p(x_k) = y_k$ and $p'(x_k) = \dot{y}_k$. Here $p'(x)$ denotes the 1st-order derivative of $p(x)$.

The key of this implementation is how to compute $\dot{y}_k$ such that the interpolation is accurate and $p(x)$ is non-decreasing. The value of $\dot{y}_k$ is decided by two steps. First, we compute the first-order derivative $\dot{y}(x_k)$ by a parabolic method:

$$\dot{y}(x_k) = \begin{cases} \dfrac{s_1(2\Delta x_1 + \Delta x_2) - s_2\Delta x_1}{x_3 - x_1}, & \text{if } k = 1 \\ \dfrac{s_{n-1}(2\Delta x_{n-1} + \Delta x_{n-2}) - s_{n-2}\Delta x_{n-1}}{x_n - x_{n-2}}, & \text{if } k = n \\ \dfrac{s_k\Delta x_{k-1} + s_{k-1}\Delta x_k}{x_{k+1} - x_{k-1}}, & \text{if } 2 < k < n-1. \end{cases} \tag{7.8}$$

Algorithm 4 piecewise cubic density estimation

1: Evaluate the model (7.1) to obtain N samples of $\hat{x}$;
2: Shift and scale $\hat{x}$ to obtain N samples for x;
3: Pick n data points (x_k, y_k), under constraint (7.4);
4: Calculate $\dot{y}(x_k)$ using the parabolic method (7.8);
5: **for** $k = 1, \cdots, n$ **do**
6: **if** $y_k = y_{k+1}$, set $c_k^1 = y_k$ and $c_k^2 = c_k^3 = c_k^4 = 0$;
7: **else**
8: Compute $\dot{y}_k$ according to (7.9);
9: Compute the coefficients in (7.7).
10: **end**
11: **end for**

This parabolic method has a 2nd-order accuracy [157]. Second, $\dot{y}_k$ is obtained by perturbing $\dot{y}(x_k)$ (if necessary) to enforce the monotonicity of $p(x)$. The monotonicity of $p(x)$ is equivalent to $p'(x) \geq 0$, which is a 2nd-order inequality. By solving this inequality, a feasible region for $\dot{y}_k$, denoted by $\mathcal{A}$, is provided in [156]. Occasionally we need to project $\dot{y}(x_k)$ onto $\mathcal{A}$ to get $\dot{y}_k$ if $\dot{y}(x_k) \notin \mathcal{A}$. In practice, we use the simpler projection method suggested by [157]:

$$\dot{y}_k = \begin{cases} \min\left(\max\left(0, \dot{y}(x_k)\right), 3s_{\min}^k\right), & \text{if } s_k s_{k-1} > 0 \\ 0, & \text{if } s_k s_{k-1} = 0 \end{cases} \tag{7.9}$$

with $s_0 = s_1$, $s_n = s_{n-1}$ and $s_{\min}^k = \min(s_k, s_{k-1})$. The above procedure projects $\dot{y}(x_k)$ onto a subset of $\mathcal{A}$, and thus the monotonicity of $p(x)$ is guaranteed.

Once $p(x)$ is constructed, the probability density function of x can be obtained by

$$\rho(x) = p'(x) = c_k^2 + 2c_k^3(x - x_k) + 3c_k^4(x - x_k)^2 \tag{7.10}$$

for $x_k \leq x \leq x_{k+1}$. Note that for $x \notin [x_1, x_n]$, $p'(x) = 0$.

Calculating $p'(x)$ may amplify the interpolation errors. However, the error is acceptable since the constructed $p(x)$ is smooth enough and $p'(x)$ is continuous. The pseudo codes of Algorithm 4 summarize the steps of this approach.

7.2.2 Method 2: Piecewise Rational Quadratic Interpolation

Our second implementation is based on a piecewise rational quadratic interpolation [158, 159]. In this implementation, we approximate the cumulative density function of x by

$$p(x) = \frac{N(x)}{D(x)} = \frac{\alpha_k^1 + \alpha_k^2 x + \alpha_k^3 x^2}{\beta_k^1 + \beta_k^2 x + \beta_k^3 x^2} \tag{7.11}$$

for $x \in [x_k, x_{k+1}]$. The coefficients are selected by the following method: when $x_k = x_{k+1}$, we set $\alpha_k^1 = y_k$, $\beta_k^1 = 1$ and all other coefficients to zero; otherwise, the coefficients are decided according to the formula

$$\begin{aligned}
&\alpha_k^1 = y_{k+1}x_k^2 - w_k x_k x_{k+1} + y_k x_{k+1}^2, \\
&\alpha_k^2 = w_k(x_k + x_{k+1}) - 2y_{k+1}x_k - 2y_k x_{k+1},\ \alpha_k^3 = y_{k+1} - w_k + y_k, \\
&\beta_k^1 = x_k^2 - v_k x_k x_{k+1} + x_{k+1}^2,\ \beta_k^2 = v_k(x_k + x_{k+1}) - 2x_k - 2x_{k+1},\ \beta_k^3 = 2 - v_k, \\
&\text{with } w_k = \frac{y_{k+1}\dot{y}_k + y_k\dot{y}_{k+1}}{s_k} \text{ and } v_k = \frac{\dot{y}_k + \dot{y}_{k+1}}{s_k}
\end{aligned} \tag{7.12}$$

where s_k is defined the same as in piecewise cubic interpolation. In this interpolation scheme, the sufficient and necessary condition for the monotonicity of $p(x)$ is very simple: $\dot{y}_k \geq 0$. In order to satisfy this requirement, the slope $\dot{y}_k$ is approximated by the geometric mean

$$\dot{y}_k = \begin{cases} (s_1)^{\frac{x_3 - x_1}{x_3 - x_2}} (s_{3,1})^{\frac{x_1 - x_2}{x_3 - x_2}}, & \text{if } k = 1 \\ (s_{n-1})^{\frac{x_n - x_{n-2}}{x_{n-1} - x_{n-2}}} (s_{n,n-2})^{\frac{x_{n-1} - x_n}{x_{n-1} - x_{n-2}}}, & \text{if } k = n \\ (s_{k-1})^{\frac{x_{k+1} - x_k}{x_{k+1} - x_{k-1}}} (s_k)^{\frac{x_k - x_{k-1}}{x_{k+1} - x_{k-1}}}, & \text{if } 1 < k < n \end{cases} \tag{7.13}$$

with $s_{k_1,k_2} = \frac{y_{k_1} - y_{k_2}}{x_{k_1} - x_{k_2}}$. Similarly, the probability density function of x can be approximated by

$$\rho(x) = p'(x) = \frac{N'(x)D(x) - D'(x)N(x)}{D^2(x)}, \tag{7.14}$$

for $x \in [x_k, x_{k+1}]$.

Note that in piecewise cubic interpolation, a projection procedure is not required, since the monotonicity of $p(x)$ is automatically guaranteed. The pseudo codes of this

Algorithm 5 piecewise rational quadratic density estimation

1: Evaluate the model (7.1) to obtain N samples of x;
2: Shift and scale $\hat{x}$ to obtain N samples for x;
3: Pick n data points (x_k, y_k), under constraint (7.4);
4: **for** $k = 1, \cdots, n$ **do**
5: Calculate $\dot{y}_k$ using the formula in (7.13);
6: **if** $y_k = y_{k+1}$
7: set $\alpha_k^1 = y_k$, $\beta_k^1 = 1$ and other coefficients to zero;
8: **else**
9: compute the coefficients of $N(x)$ and $D(x)$ using (7.12).
10: **end**
11: **end for**

density estimation method are provided in Algorithm 5.

7.2.3 Properties of $p(x)$

It is straightforward to show that the obtained density functions are physically consistent: 1) $p(x)$ is differentiable, and thus its derivative $p'(x)$ always exists; 2) $p(x)$ is monotonically increasing from 0 to 1, and the probability density function $\rho(x)$ is non-negative.

We can easily draw a random sample from the obtained $p(x)$. Let $y \in [0, 1]$ be a sample from a uniform distribution, then a sample of x can be obtained by solving $p(x) = y$ in the interval $y \in [y_k, y_{k+1}]$. This procedure only requires computing the roots of a cubic (or quadratic) polynomials, resulting in a unique solution $x \in [x_k, x_{k+1}]$. This property is very useful in uncertainty quantification. Not only are random samples used in Monte Carlo simulators, but also they can be used in stochastic spectral methods. Recently, compressed sensing has been applied to high-dimensional stochastic problems [88,89,164]. In compressed sensing, random samples are normally used to enhance the restricted isometry property of the dictionary matrix [92].

Finally, it becomes easy to determine the generalized polynomial-chaos basis functions and a proper quadrature rule for x due to the special form of $\rho(x)$. This issue will be discussed in Section 7.3.

Remark 4.1: Our proposed density estimator only requires some interpolation points from a discrete cumulative density function curve. The interpolation points actually can be obtained by any appropriate approach. For example, kernel density estimation will be a good choice if we know a proper kernel function and a good smoothing parameter based on *a-priori* knowledge. When the surrogate model is a linear quadratic function of Gaussian variables, we may first employ asymptotic probability extraction [2] to generate a physically inconsistent cumulative density function. After that, some monotone data points (with y_i's bounded by 0 and 1) can be selected to generate a piecewise cubic or piecewise rational quadratic cumulative density function. The new cumulative density function and probability density function become physically consistent and can be reused in a stochastic simulator.

7.3 Determine Basis Functions and Gauss Quadrature Rules

This section shows how to calculate the generalized polynomial-chaos bases and the Gauss quadrature points/weights of x based on the obtained density function.

7.3.1 Proposed Implementation

One of the many usages of our density estimator is to fast compute a set of generalized polynomial-chaos basis functions and Gauss quadrature points/weights by analytically computing the integrals in (2.4). Let $\pi_i^2(x) = \sum_{k=0}^{2i} \tau_{i,k} x^k$, then we have

$$\int_{\mathbb{R}} x\pi_i^2(x)\rho(x)dx = \sum_{k=0}^{2i} \tau_{i,k} M_{k+1}, \int_{\mathbb{R}} \pi_i^2(x)\rho(x)dx = \sum_{k=0}^{2i} \tau_{i,k} M_k \tag{7.15}$$

where M_k denotes the k-th statistical moments of x. By exploiting the special form of our obtained density function, the statistical moments can be computed as

$$M_k = \int_{-\infty}^{+\infty} x^k \rho(x) dx = \int_{x_1}^{x_n} x^k \rho(x) dx = \sum_{j=1}^{n-1} I_{j,k} \tag{7.16}$$

where $I_{j,k}$ denotes the integral in the j-th piece:

$$I_{j,k} = \int_{x_j}^{x_{j+1}} x^k \rho(x) dx = F_{j,k}(x_{j+1}) - F_{j,k}(x_j). \tag{7.17}$$

Here $F_{j,k}(x)$ is a continuous analytical function under the constraint $\frac{d}{dt} F_{j,k}(x) = x^k \rho(x)$ for $x \in [x_j, x_{j+1}]$. The key problem of our method is to construct $F_{j,k}(x)$. When $\rho(x)$ is obtained from Alg. 4 or Alg. alg:mprq, we can easily obtain the closed form of $F_{j,k}(x)$, as will be elaborated in Section 7.3.2 and Section 7.3.3.

Remark 5.1: This paper directly applies (2.4) to compute the recurrence parameters γ_i and κ_i. As suggested by [46], modified Chebyshev algorithm [168] can improve the numerical stability when constructing high-order polynomials. Modified Chebyshev algorithm indirectly computes γ_i and κ_i by first evaluating a set of modified moments. Again, if we employ the $\rho(x)$ obtained from our proposed density estimators, then the calculation of modified moments can also be done analytically to further improve the accuracy and numerical stability.

7.3.2 Construct $F_{j,k}(x)$ using the Density Function from Alg. 4

When $\rho(x)$ is constructed by Alg. 4, $x^k \rho(x)$ is a polynomial function of at most degree $k+2$ inside the interval $[x_j, x_{j+1}]$. Therefore, the analytical form of $F_{j,k}(x)$ is

$$F_{j,k}(x) = \mathbf{a}_{j,k} x^{k+3} + \mathbf{b}_{j,k} x^{k+2} + \mathbf{c}_{j,k} x^{k+1} \tag{7.18}$$

with

$$\mathbf{a}_{j,k} = \tfrac{3c_j^4}{k+3}, \quad \mathbf{b}_{j,k} = \tfrac{2c_j^3 - 6c_j^4 x_j}{k+2}, \quad \mathbf{c}_{j,k} = \tfrac{c_j^2 - 2c_j^3 x_j + 3c_j^4 x_j^2}{k+1}.$$

7.3.3 Construct $F_{j,k}(x)$ using the Density Function from Alg. 5

If $\rho(x)$ is constructed by Alg. 5, for any $x \in [x_j, x_{j+1}]$ we rewrite $x^k\rho(x)$ as follows

$$x^k\rho(x) = \frac{x^k[N'(x)D(x)-D'(x)N(x)]}{D^2(x)} = \frac{d}{dx}\left(\frac{x^kN(x)}{D(x)}\right) - \frac{kx^{k-1}N(x)}{D(x)}.$$

Therefore, $F_{j,k}(x)$ can be selected as

$$F_{j,k}(x) = \frac{x^kN(x)}{D(x)} - \tilde{F}_{j,k}(x), \text{ with } \frac{d}{dx}\tilde{F}_{j,k}(x) = \frac{kx^{k-1}N(x)}{D(x)}.$$

In order to obtain $\tilde{F}_{j,k}(x)$, we perform a long division:

$$\frac{kx^{k-1}N(x)}{D(x)} = \tilde{P}_{j,k}(x) + \frac{\tilde{R}_{j,k}(x)}{D(x)} \tag{7.19}$$

where $\tilde{P}_{j,k}(x)$ and $\tilde{R}_{j,k}(x)$ are both polynomial functions, and $\tilde{R}_{j,k}(x)$ has a lower degree than $D(x)$. Consequently,

$$\tilde{F}_{j,k}(x) = \tilde{F}^1_{j,k}(x) + \tilde{F}^2_{j,k}(x) \tag{7.20}$$

where $\tilde{F}^1_{j,k}(x)$ and $\tilde{F}^2_{j,k}(x)$ are the integrals of $\tilde{P}_{j,k}(x)$ and $\frac{\tilde{R}_{j,k}(x)}{D(x)}$, respectively. It is trivial to obtain $\tilde{F}^1_{j,k}(x)$ since $\tilde{P}_{j,k}(x)$ is a polynomial function.

The closed form of $\tilde{F}^2_{j,k}(x)$ is decided according to the coefficients of $D(x)$ and $\tilde{R}_{j,k}(x)$, as is summarized below.

Case 1: if $\beta^3_j \neq 0$, then $\tilde{R}_{j,k}(x) = \tilde{r}^0_{j,k} + \tilde{r}^1_{j,k}x$. Let us define $\Delta_j := 4\beta^1_j\beta^3_j - \beta^2_j$, then we can select $\tilde{F}^2_{j,k}(x)$ according to the formula in (7.21).

$$\tilde{F}^2_{j,k}(x) = \begin{cases} \frac{\tilde{r}^1_{j,k}}{2\beta^3_j}\ln\left|\beta^3_jx^2+\beta^2_jx+\beta^1_j\right| + \frac{2\beta^3_j\tilde{r}^0_{j,k}-\beta^2_j\tilde{r}^1_{j,k}}{\beta^3_j\sqrt{\Delta_j}}\arctan\frac{2\beta^3_jx+\beta^2_j}{\sqrt{\Delta_j}}, & \text{if } \Delta_j > 0 \\ \frac{\tilde{r}^1_{j,k}}{2\beta^3_j}\ln\left|\beta^3_jx^2+\beta^2_jx+\beta^1_j\right| - \frac{2\beta^3_j\tilde{r}^0_{j,k}-\beta^2_j\tilde{r}^1_{j,k}}{\beta^3_j\sqrt{-\Delta_j}}\arctan\frac{2\beta^3_jx+\beta^2_j}{\sqrt{-\Delta_j}}, & \text{if } \Delta_j < 0 \\ \frac{\tilde{r}^1_{j,k}}{2\beta^3_j}\ln\left|\beta^3_jx^2+\beta^2_jx+\beta^1_j\right| - \frac{2\beta^3_j\tilde{r}^0_{j,k}-\beta^2_j\tilde{r}^1_{j,k}}{\beta^3_j\left(2\beta^3_jx+\beta^2_j\right)}, & \text{if } \Delta_j = 0 \end{cases} \tag{7.21}$$

Case 2: if $\beta^3_j = 0$ and $\beta^2_j \neq 0$, then $\tilde{R}_{j,k}(x) = \tilde{r}^0_{j,k}$ is a constant. In this case, we

select

$$\tilde{F}^2_{j,k}(x) = \frac{\tilde{r}^0_{j,k}}{\beta^2_j} \ln\left|\beta^2_j x + \beta^1_j\right|. \tag{7.22}$$

Case 3: if $\beta^3_j = \beta^2_j = 0$, then $\tilde{R}_{j,k}(x) = 0$. In this case we set $\tilde{F}^2_{j,k}(x) = 0$.

Remark 5.2: Occasionally, the projection procedure (7.9) in Alg. 4 may cause extra errors at the end points of some intervals. If this problem happens we recommend to use Alg. 5. On the other hand, if high-order basis functions is required we recommend Alg. 4, since the moment computation with the density from Alg. 5 is numerically less stable (due to the long-term division and the operations in (7.21).

7.4 Numerical Examples

This section presents the numerical results on a synthetic example and the statistical surrogate models from two practical analog/RF circuits. The surrogate models of these practical circuits are extracted from transistor-level simulation using the fast stochastic circuit simulator developed in [34–36]. All experiments are run in Matlab on a 2.4GHz 4-GB RAM laptop.

In the following experiments, we use the density functions from kernel density estimation as the "reference solution" because: 1) as a standard technique, kernel density estimation is most widely used in mathematics and engineering; 2) kernel density estimation guarantees that the generated probability density function is non-negative, whereas asymptotic probability extraction cannot; 3) Gaussian kernel function seems to be a good choice for the examples in this paper. However, it is worth noting that the density functions from kernel density estimation are not efficient for reuse in higher-level stochastic simulation. We plot the density functions of $\hat{x}$ (the original random input) instead of x (the new random input after a linear transformation) since the original one is physically more intuitive. In order to verify the accuracy of the computed generalized polynomial-chaos bases and Gauss quadrature points/weights,

we define a symmetric matrix $\mathbf{V}_{\hat{n}+1} \in \mathbb{R}^{(\hat{n}+1)\times(\hat{n}+1)}$, the (i,j) entry of which is

$$v_{i,j} = \sum_{k=1}^{\hat{n}+1} w^k \phi_{i-1}\left(x^k\right) \phi_{j-1}\left(x^k\right).$$

Here x^k and and w^k are the computed k-th Gauss quadrature point and weight, respectively. Therefore $v_{i,j}$ approximates the inner product of $\phi_{i-1}(x)$ and $\phi_{j-1}(x)$, defined as $\int_{\mathbb{R}} \phi_{i-1}(x)\,\phi_{j-1}(x)\,\rho(x)dx$, by $\hat{n}+1$ quadrature points. Let $\mathbf{I}_{\hat{n}+1}$ be an identity matrix, then we define an error:

$$\epsilon = ||\mathbf{I}_{\hat{n}+1} - \mathbf{V}_{\hat{n}+1}||_\infty \tag{7.23}$$

which is close to zero when our constructed basis functions and Gauss-quadrature points/weights are accurate enough.

7.4.1 Synthetic Example

As a demonstration, we first consider the following synthetic example with four random parameters $\vec{\xi} = [\xi_1, \cdots, \xi_4]$:

$$\hat{x} = f(\vec{\xi}) = \xi_1 + 5\exp(0.52\xi_2) + 0.3\sqrt{2.1 \times |\xi_4|} + \sin(\xi_3)\cos(3.91\xi_4)$$

where ξ_1, ξ_2 and ξ_3 are all standard Gaussian random variables, and ξ_4 has a uniform distribution in the interval $[-0.5, 0.5]$. This model is strongly nonlinear with respect to $\vec{\xi}$ due to the exponential, triangular and square root functions. It is also non-smooth at $\xi_4 = 0$ due to the third term in the model. This model is designed to challenge our algorithm. Using this surrogate model, 10^6 samples of x are easily created to generate the cumulative density function curve within 1 second.

Density Estimation: we set $m = 45$ and select 74 data points from the obtained cumulative density function curve using the constraint in (7.5). After that, both Alg. 4 and Alg. 5 are applied to generate $p(x)$ and $\rho(x)$ as approximations to the cumulative density function and probability density function of x, respectively. The

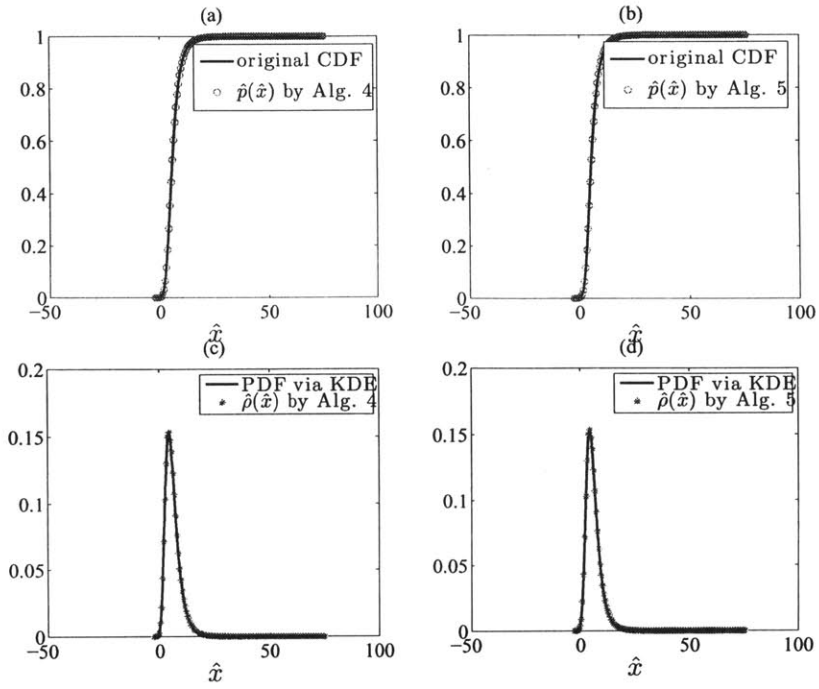


Figure 7-2: Cumulative density function (CDF) and probability density function (PDF) approximation of $\hat{x}$ for the synthetic example. The reference PDF is generated by kernel density estimation (KDE).

CPU times cost by our proposed density estimators are in millisecond scale, since only simple algebraic operations are required. After scaling by (7.3), the cumulative density function and probability density function of the original random input $\hat{x}$ ($\hat{p}(\hat{x})$ and $\hat{\rho}(\hat{x})$, respectively) from both algorithms are compared with the original cumulative density function and probability density function in Fig. 7-2. Clearly, $\hat{p}(\hat{x})$ is indistinguishable with the original cumulative density function (from Monte Carlo simulation); and $\hat{\rho}(\hat{x})$ overlaps with the original probability density function (estimated by kernel density estimation using Gaussian kernels). Note that the results from kernel density estimation are not efficient for reuse in higher-level stochastic simulation, since all Monte Carlo samples are used as parameters of the resulting density function.

It is clearly shown that the generated $\hat{p}(\hat{x})$ [and thus $p(x)$] is monotonically increasing from 0 to 1, and that the generated $\hat{\rho}(\hat{x})$ [and thus $\rho(x)$] is non-negative.

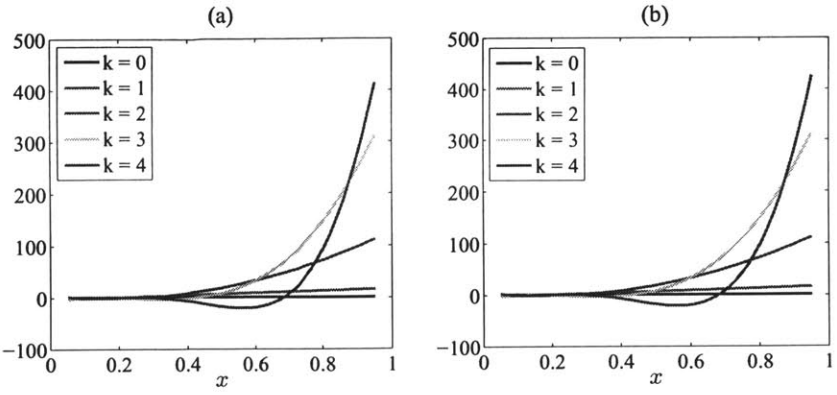


Figure 7-3: Computed generalized polynomial-chaos basis functions $\phi_k(x)$ $(k = 0, \cdots, 4)$ for the synthetic example. (a) uses the probability density function from Alg. 4, and (b) uses the probability density function from Alg. 5.

Table 7.1: Computed Gauss quadrature points and weights for the synthetic example.

with $\rho(x)$ from Alg. 4		with $\rho(x)$ from Alg. 5	
x^k	w^k	x^k	w^k
0.082620	0.311811	0.084055	0.332478
0.142565	0.589727	0.144718	0.576328
0.249409	0.096115	0.252980	0.089027
0.458799	0.002333	0.463207	0.002150
0.837187	0.000016	0.835698	0.000016

Therefore, the obtained density functions are physically consistent.

Basis Function: Using the obtained density functions and the proposed implementation in Section 7.3, a set of orthonormal polynomials $\phi_k(x)$'s are constructed as the basis functions at the cost of milliseconds. Fig. 7-3 show the first five generalized polynomial-chaos basis functions. Note that although the computed basis functions from two methods are graphically indistinguishable, they are actually slightly different since Alg. 4 and Alg. 5 generate different representations for $\rho(x)$.

Gauss Quadrature Rule: setting $\hat{n} = 4$, five Gauss quadrature points and weights are generated using the method presented in Section 7.3. Table 7.1 shows the results from two kinds of approximated density functions. Clearly, since the probability density functions from Alg. 4 and Alg. 5 are different, the resulting quadrature points/weights are also slightly different. The results from both probability density

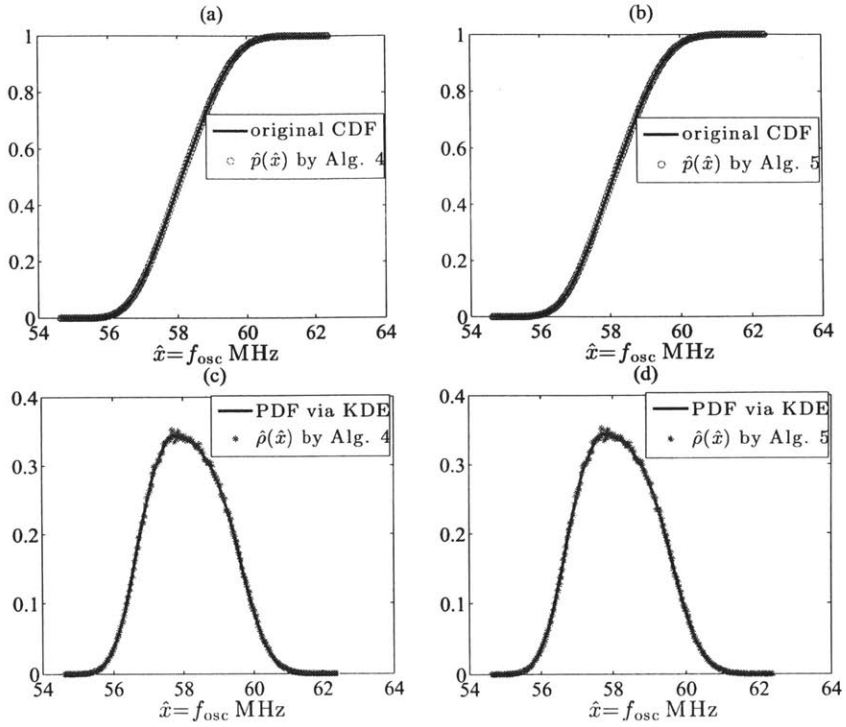


Figure 7-4: Cumulative density function (CDF) and probability density function (PDF) approximation for the frequency of the Colpitts oscillator. The reference PDF is generated by kernel density estimation (KDE).

functions are very accurate. Using the probability density function from Alg. 4, we have $\epsilon = 2.24 \times 10^{-14}$, and the error (7.23) is 7.57×10^{-15} if $\rho(x)$ from Alg. 5 is employed.

7.4.2 Colpitts Oscillator

We now test our proposed algorithm on a more practical example, the Colpitts oscillator circuit shown in Fig. 5-3. In this circuit, L_1=150 + $\mathcal{N}(0, 9)$ nH and C_1=100 + $\mathcal{U}(-10, 10)$ pF are random variables with Gaussian and uniform distributions, respectively. We construct a surrogate model using generalized polynomial chaos expansions and the stochastic shooting Newton solver in [36]. The oscillation frequency $f_{\rm osc}$ is

expressed as

$$\hat{x} = f_{\text{osc}} = f(\vec{\xi}) = \frac{1}{\sum_{k=1}^{10} T_k \psi_k(\vec{\xi})} \tag{7.24}$$

where the denominator is a 3rd-order generalized polynomial chaos representation for the period of the oscillator, with $\psi_k(\vec{\xi})$ being the k-th multivariate generalized polynomial-chaos basis function of $\vec{\xi}$ and T_k the corresponding coefficient. Although the period is a polynomial function of $\vec{\xi}$, the frequency is not, due to the inverse operation. In order to extract the cumulative density function curve, 5×10^5 samples are utilized to evaluate the surrogate model (7.24) by Monte Carlo, which costs 225 seconds of CPU times on our Matlab platform.

Density Estimation: 106 data points on the obtained cumulative density function curve are used to construct $p(x)$ and $\rho(x)$, which costs only several milliseconds. After scaling the constructed closed-form cumulative density functions and probability density functions from Alg. 4 and Alg. 5, the approximated density functions of the oscillation frequency are compared with the Monte Carlo results in Fig. 7-4. The constructed cumulative density functions by both methods are graphically indistinguishable with the result from Monte Carlo. The bottom plots in Fig. 7-4 also show a good match between our obtained $\hat{\rho}(\hat{x})$ with the result from kernel density estimation. Again, important properties of the density functions (i.e., monotonicity and boundedness of the cumulative density function, and non-negativeness of the probability density function) are well preserved by our proposed density estimation algorithms.

Basis Function: Using the obtained density functions and the proposed implementation in Section 7.3, a set of orthonormal polynomials $\phi_k(x)$'s are constructed as the basis functions at the cost of milliseconds. Fig. 7-5 shows several generalized polynomial-chaos basis functions of x. Again, the basis functions resulting from our two density estimation implementations are only slightly different.

Gauss Quadrature Rule: the computed five Gauss quadrature points and weights are shown in Table 7.2. Again the results from two density estimations are slightly different. The results from both probability density functions are very accurate. Using $\rho(x)$ from Alg. 4, we have $\epsilon = 1.3 \times 10^{-13}$, and the error is 1.45×10^{-13} if we use $\rho(x)$

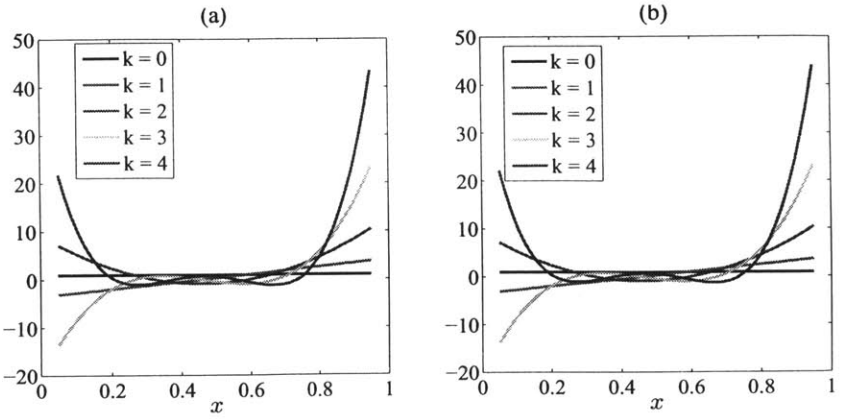


Figure 7-5: Computed generalized polynomial-chaos basis functions $\phi_k(x)$ ($k = 0, \cdots, 4$) for the Colpitts oscillator. (a) uses the probability density function from Alg. 4, and (b) uses the probability density function from Alg. 5.

Table 7.2: Computed Gauss quadrature points and weights for the Colpitts oscillator.

with $\rho(x)$ from Alg. 4		with $\rho(x)$ from Alg. 5	
x^k	w^k	x^k	w^k
0.170086	0.032910	0.170935	0.032456
0.309764	0.293256	0.310016	0.292640
0.469034	0.441303	0.468658	0.439710
0.632232	0.217359	0.631249	0.218274
0.788035	0.016171	0.786226	0.016820

from Alg. 5.

7.4.3 Low-Noise Amplifier

In this example we consider the statistical behavior of the total harmonic distortion at the output node of the low-noise amplifier shown in Fig. 4-7. The device ratios of the MOSFETs are W_1/L_1=W_2/L_2=500/0.35 and W_3/L_3=50/0.35. The linear components are R_1=50Ω, R_2=2 kΩ, C_1=10 pF, C_L=0.5 pF, L_1=20 nH and L_3=7 nH. Four random parameters are introduced to describe the uncertainties: ξ_1 and ξ_2 are standard Gaussian variables, ξ_3 and ξ_4 are standard uniform-distribution parameters. These random parameters are mapped to the physical parameters as follows: temperature T=$300 + 40\xi_1$ K influences transistor threshold voltage; V_T=$0.4238 + 0.1\xi_2$ V

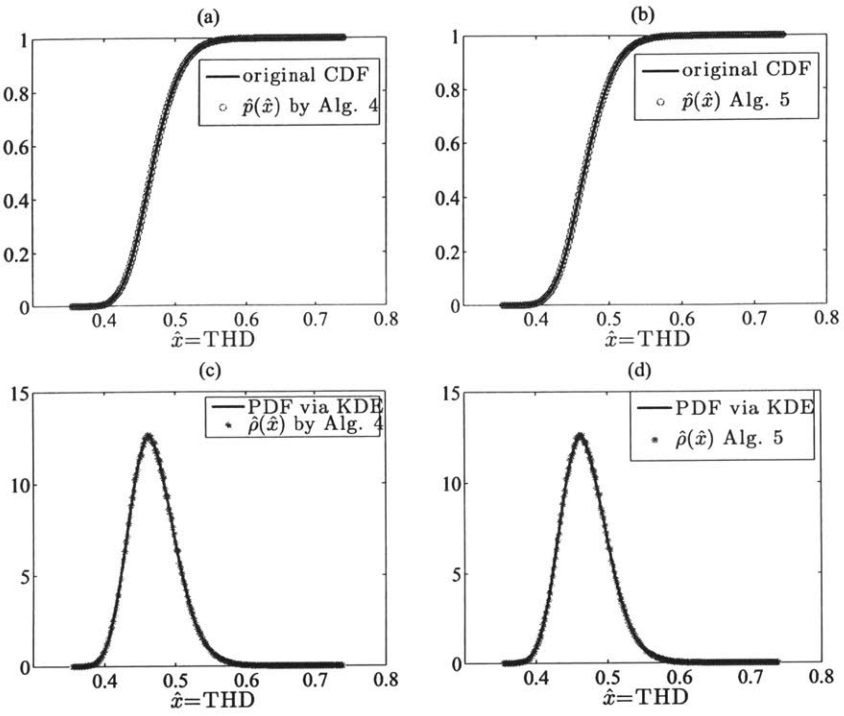


Figure 7-6: Cumulative density function (CDF) and probability density function (PDF) for the total harmonic distortion (THD) of the low-noise amplifier. The reference PDF is generated by kernel density estimation (KDE).

represents the threshold voltage under zero V_{bs}; R_3=0.9+0.2ξ_3 kΩ and L_2=0.8+1.2ξ_4 nH. The supply voltage is V_{dd}=1.5 V, and the periodic input is $V_{\text{in}} = 0.1\sin(4\pi \times 10^8 t)$ V.

The surrogate model for total harmonic distortion analysis is constructed by a numerical scheme as follows. First, the parameter-dependent periodic steady-state solution at the output is solved by the non-Monte Carlo simulator in [36], and is expressed by a truncated generalized polynomial chaos representation with K basis functions:

$$V_{\text{out}}(\vec{\xi}, t) = \sum_{k=1}^{K} v_k(t)\psi_k(\vec{\xi})$$

where $v_k(t)$ is the time-dependent coefficient of the generalized polynomial chaos expansion for the periodic steady-state solution and is actually solved at a set of time points during the entire period $[0, T]$. Next, $v_k(t)$ is expressed by a truncated Fourier

series:

$$v_k(t) = \frac{a_k^0}{2} + \sum_{j=1}^{J} \left(a_k^j \cos(j\omega t) + b_k^j \sin(j\omega t) \right)$$

with $\omega = \frac{2\pi}{T}$. The coefficients a_k^j and b_k^j

$$a_k^j = \frac{2}{T} \int_0^T v_k(t) \cos(j\omega t) dt, \;\; b_k^j = \frac{2}{T} \int_0^T v_k(t) \sin(j\omega t) dt$$

are computed by a Trapezoidal integration along the time axis. Finally, the parameter-dependent total harmonic distortion is obtained as

$$\begin{aligned} \hat{x} = \text{THD} = f(\vec{\xi}) &= \sqrt{\frac{\sum_{j=2}^{J} \left[\left(a^j(\vec{\xi}) \right)^2 + \left(b^j(\vec{\xi}) \right)^2 \right]}{\left(a^1(\vec{\xi}) \right)^2 + \left(b^1(\vec{\xi}) \right)^2}} \\ \text{with } a^j(\vec{\xi}) = \sum_{k=1}^{K} a_k^j \phi_k(\vec{\xi}), \;\; b^j(\vec{\xi}) &= \sum_{k=1}^{K} a_k^j \phi_k(\vec{\xi}). \end{aligned} \tag{7.25}$$

We set $J = 5$ in the Fourier expansion, which is accurate enough for this low-noise amplifier. We use a 3rd-order generalized polynomial chaos expansion, leading to K=35. This surrogate model is evaluated by Monte Carlo with 5×10^5 samples at the cost of 330 seconds.

Density Estimation: 114 points are selected from the obtained cumulative density function curve to generate $p(x)$ and $\rho(x)$ by Alg. 4 and Alg. 5, respectively, which costs only several milliseconds. After scaling, Fig. 7-6 shows the closed-form density functions for the total harmonic distortion of this low-noise amplifier, which matches the results from Monte Carlo simulation very well. The generated $p(x)$ monotonically increases from 0 to 1, and $\rho(x)$ is non-negative. Therefore, the obtained density functions are physically consistent.

Basis Function: Using the obtained density functions, several orthonormal polynomials of x are constructed. Fig. 7-7 shows the first five basis functions of x. Again, the basis functions resulting from our two density estimation implementations look similar since the density functions from both methods are only slightly different.

Gauss Quadrature Rule: Five Gauss quadrature points and weights are computed

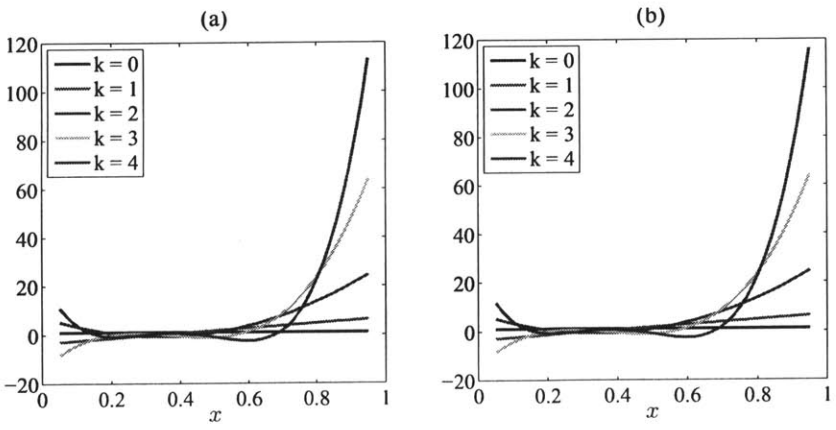


Figure 7-7: Computed generalized polynomial-chaos basis functions $\phi_k(x)$ $(k = 0, \cdots, 4)$ for the low-noise amplifier. (a) uses the probability density function from Alg. 4, and (b) uses the probability density function from Alg. 5.

Table 7.3: Computed Gauss quadrature points and weights for the low-noise amplifier.

with $\rho(x)$ from Alg. 4		with $\rho(x)$ from Alg. 5	
x^k	w^k	x^k	w^k
0.131542	0.056766	0.140381	0.073309
0.251826	0.442773	0.261373	0.470691
0.385311	0.4432588	0.395704	0.400100
0.550101	0.066816	0.561873	0.055096
0.785055	0.001056	0.798122	0.000803

and listed in Table 7.3. Again the results from two density estimations are slightly different due to the employment of different density estimators. When the density functions from piecewise cubic and piecewise rational quadratic interpolations are used, the the errors defined in (7.23) are 3.11×10^{-14} and 4.34×10^{-14}, respectively.

7.4.4 Comparison with Asymptotic Probability Extraction

Finally we test our examples by the previous asymptotic probability extraction algorithm [2,3]. Since our surrogate models are not in linear quadratic forms, we slightly modify asymptotic probability extraction: as done in [169] we use Monte Carlo to compute the statistical moments. All other procedures are exactly the same with those in [2,3].

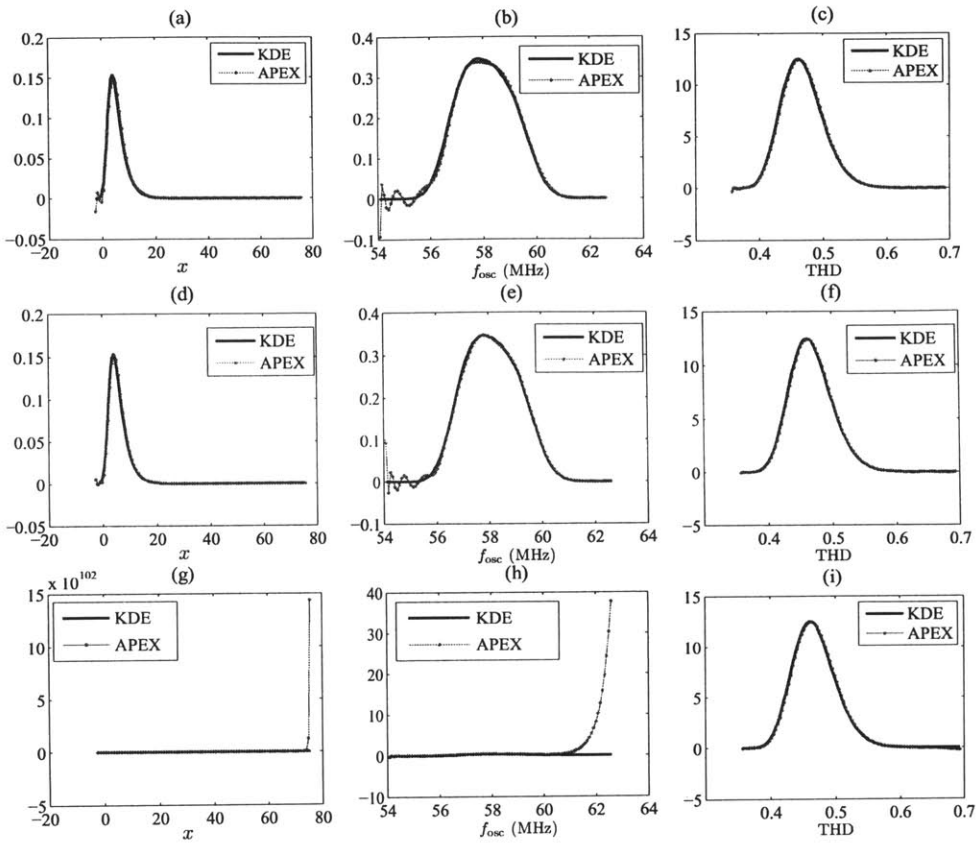


Figure 7-8: Probability density functions extracted by asymptotic probability extraction (APEX) [2,3], compared with the results from kernel density estimation (KDE). Left column: the synthetic example. Central column: frequency of the Colpitts oscillator. Right column: total harmonic distortion (THD) of the low-noise amplifier. (a)-(c): with 10 moments; (d)-(f): with 15 moments; (g)-(i): with 17 moments.

As shown in Fig. 7-8, asymptotic probability extraction produces some negative probability density function values for the synthetic example and the Colpitts oscillator. The probability density functions of the low-noise amplifier are also slightly below 0 in the tail regions, which is not clearly visible in the plots. Compared with the results from our proposed algorithms (that are non-negative and graphically indistinguishable with the original probability density functions), the results from asymptotic probability extraction have larger errors. As suggested by [2,3], we increase the order of moment matching to 15, hoping to produce non-negative results. Unfortunately, Fig. 7-8 (d) and (e) show that negative probability density function values still ap-

pear, although the accuracy is improved around the peaks. Further increasing the order to 17, we observe that some positive poles are generated by asymptotic waveform evaluation [170]. Such positive poles make the computed probability density functions unbounded and far from the original ones, as demonstrated by Fig. 7-8 (g) & (h). For the low-noise amplifier, the approximated probability density function curve also becomes unbounded once we increase the order of moment matching to 20, which is not shown in the plot.

These undesirable phenomenon of asymptotic probability extraction is due to inexact moment computation and the inherent numerical instability of asymptotic waveform evaluation [170]. Although it is possible to compute the statistical moments in some other ways (e.g., using maximum likelihood [82] or point estimation method [139]), the shortcomings of asymptotic waveform evaluation (i.e., numerical instability and causing negative impulse response for a linear system) cannot be overcome. Because the density functions from asymptotic probability extraction may be physically inconsistent, they cannot be reused in a stochastic simulator (otherwise non-physical results may be obtained). Since the obtained probability density function is not guaranteed non-negative, the computed κ_i in the three-term relation (2.4) may become negative, whereas (2.4) implies that κ_i should always be non-negative.

7.5 Limitations and Possible Solutions

7.5.1 Lack of Accuracy for Tail Approximation

In some applications (e.g., SRAM cell design), users require a highly accurate description about the density function in the tail region. Our algorithm does not work for such applications because of the following reasons.

1. In order to approximate the tail region, a lot of samples should be drawn, leading to a high computational cost.

2. Our algorithm does not provide enough accuracy. In SRAM analysis, it is very common that the estimated failure probability should be below 10^{-6}. Such high

accuracy cannot be reached by our interpolation-based algorithms.

3. Our algorithm uses a surrogate model to draw samples, the accuracy of a surrogate model is typically not enough for tail analysis.

7.5.2 Multiple Correlated Outputs of Interests

The proposed algorithm is designed to approximate a scalar output of interest. When multiple correlated outputs of interest are required for high-level uncertainty quantification, we may need to calculate a joint density function which cannot be easily captured by the proposed piecewise interpolation. In order to solve this problem, other density estimation techniques may be exploited, such as maximum entropy or Gaussian mixture modeling.

Chapter 8

Conclusions and Future Work

8.1 Summary of Results

In this thesis, we have developed a set of algorithms to efficiently quantify the parametric uncertainties in nano-scale integrated circuits and microelectromechanical systems (MEMS). Our algorithms have shown significant speedup (up to $10^3\times$) over state-of-the-art circuit/MEMS simulators.

The main results of this thesis are summarized below.

Chapter 4 has developed an intrusive-type stochastic solver, named stochastic testing, to quantify the uncertainties in transistor-level circuit simulation. With generalized polynomial-chaos expansions, this simulator can handle both Gaussian and non-Gaussian variations. Compared with stochastic-collocation and stochastic-Galerkin implementations, our approach can simultaneously allow decoupled numerical simulation and adaptive step size control. In addition, multivariate integral calculation is avoided in the simulator. Such properties make the proposed method hundreds to thousands of times faster over Monte Carlo, and tens to hundreds of times faster than stochastic Galerkin. The speedup of our simulator over stochastic collocation is caused by two factors: 1) a smaller number of samples required to assemble the deterministic equation; and 2) adaptive time stepping in the intrusive stochastic testing simulator. The overall speedup factor of stochastic testing over stochastic collocation is normally case dependent. Various simulations (e.g., DC, AC and transient analysis)

have been performed on extensive analog, digital and radio-frequency (RF) circuits, demonstrating the effectiveness of our proposed algorithm.

Chapter 5 has further developed an intrusive periodic steady-state simulator for the uncertainty quantification of analog/RF circuits (including both forced circuits and oscillators). The main advantage of our proposed method is that the Jacobian can be decoupled to accelerate numerical computations. Numerical results show that our approach obtains results consistent with Monte Carlo simulation, with 2$\sim$3 orders of magnitude speedup. Our method is significantly faster over existing stochastic Galerkin-based periodic steady-state solver, and the speedup factor is expected to be more significant as the circuit size and the number of basis functions increase.

Chapter 6 has developed a hierarchical uncertainty quantification algorithm to simulate high-dimensional electronic systems. The basic idea is to perform non-Monte-Carlo uncertainty quantification at different levels. The surrogate models obtained at the low-level are used to recompute basis functions and Gauss-quadrature rules for high-level simulation. This algorithm has been demonstrated by a low-dimensional example, showing 250$\times$ speedup. A framework to accelerate the hierarchical uncertainty quantification of stochastic circuits/systems with high-dimensional subsystems has been further proposed. We have developed a sparse stochastic testing simulator based on analysis of variance (ANOVA) to accelerate the low-level simulation, and a tensor-based technique for handling high-dimensional surrogate models at the high level. Both algorithms have a linear (or near-linear) complexity with respect to the parameter dimensionality. Our simulator has been tested on an oscillator circuit with four MEMS capacitors and totally 184 random parameters, achieving highly accurate results at the cost of 10-min CPU time in MATLAB. In this example, our method is over 92$\times$ faster than the hierarchical Monte Carlo method developed in [1], and is about 14$\times$ faster than the method that uses ANOVA-based solver at the low level and Monte Carlo at the high level.

Chapter 7 has proposed an alternative framework to determine generalized polynomial-chaos basis functions and Gauss quadrature rules from possibly non-smooth surrogate models. Starting from a general surrogate model, closed-form density functions have

been constructed by two monotone interpolation techniques. It has been shown that the obtained density functions are physically consistent: the cumulative density function is monotone and bounded by 0 and 1; the probability density function is guaranteed non-negative. Such properties are not guaranteed by existing moment-matching density estimators. By exploiting the special forms of our obtained probability density functions, generalized polynomial-chaos basis functions and Gauss quadrature rules have been easily determined, which can be used for higher-level stochastic simulation. The effectiveness of our proposed algorithms has been verified by several synthetic and practical circuit examples, showing excellent efficiency (at the cost of milliseconds) and accuracy (with errors around 10^{-14}). The obtained generalized polynomial-chaos basis functions and Gauss quadrature points/weights allow standard stochastic spectral methods to efficiently handle surrogate models in a hierarchical simulator.

Some limitations of our work have been pointed out, and some possible improvements have been suggested.

8.2 Future Work

There exist a lot of topics worth further investigation. Below we summarize a few of them.

Higher Dimensionality. In stochastic spectral methods, the number of total generalized polynomial-chaos bases increases very fast as the parameter dimensionality d increases. Consequently, the computational cost becomes prohibitively expensive when d is large. It is worth exploiting the sparsity of the coefficients to reduce the complexity. Compressed sensing [92] seems effective for behavior modeling [88], but its efficiency can degrade for simulation problems (since the coefficients of different nodal voltages and/or branch currents have different sparsity pattens). A dominant singular vector method has been proposed for high-dimensional linear stochastic problems [12], yet solving the non-convex optimization is challenging for nonlinear problems. This idea may be further extended by using the concepts of tensor factorization.

Correlated Non-Gaussian Parameters. In existing literature, the random

parameters are typically assumed mutually independent, which is not valid for many practical circuits. Unlike Gaussian variables, correlated non-Gaussian parameters cannot be easily transformed to independent ones, making the generalized polynomial-chaos basis construction challenging. A theoretical method has been proposed to deal with parameters with arbitrary density functions [47], but its numerical implementation is non-trivial.

Long-Term Integration. In digital integrated circuit simulation, normally designers have to perform a long-time transient simulation. In the applied math community, it is well known that polynomial-chaos approximation can be inaccurate for tong-time integration, despite of some improvements [171].

Approximating Non-Smooth Outputs. Generalized polynomial-chaos approximation can be a good choice if the output of interest is a smooth function of the random parameters. However, in some cases the output can be a non-smooth function (e.g., the output voltages of digital logic gates). In order to approximate such outputs, one may need to partition the parameter space.

Hierarchical Uncertainty Quantification. There are lots of problems worth investigation in the direction of hierarchical uncertainty quantification. Open problems include: 1). how to extract a high-dimensional surrogate model such that the tensor rank is as small as possible (or the tensor rank is below a provided upper bound)? 2). How to perform non-Monte-Carlo hierarchical uncertainty quantification when the outputs of different blocks are correlated? 3). How to perform non-Monte-Carlo hierarchical uncertainty quantification when y_i depends on some varying variables (e.g., time and frequency)?

Optimization Under Uncertainties. In many engineering problems (e.g., circuit design, magnetic resonance imaging (MRI) scanner design), designers hope to optimize an output of interest under some uncertainties. In these cases, a forward solver can be utilized inside the loop of stochastic optimization or robust optimization to accelerate the computation. However, the resulting optimization problem may be non-convex or be of large scale.

Quantifying Other Uncertainties. Besides parametric uncertainties, other

kinds of uncertainty sources (e.g., numerical errors, model uncertainties, electrical/thermal noise) also need to be considered. How to model such uncertainties and how to quantify them still seems an open problem.

Inverse Problems. This thesis focuses on forward uncertainty quantification solvers. However, in many engineering communities inverse problems are of great interest. In semiconductor process modeling, process modeling experts have some circuit measurement data and they aim to infer the distribution of some device-level variations. In power systems, information on some power buses can be collected by sensors, and people want to calibrate the parameters of a model to better capture the behavior of a power system. Inverse problems also widely exist in biomedical fields such as magnetic resonance imaging that infer the tissue structure of a human body from the received magnetic fields. From the mathematical perspective, many inverse problems are ill-posed and large-scale and thus they are difficult to solve.

Bibliography

[1] E. Felt, S. Zanella, C. Guardiani, and A. Sangiovanni-Vincentelli, "Hierarchical statistical characterization of mixed-signal circuits using behavioral modeling," in *Proc. Intl. Conf. Computer-Aided Design.* Washington, DC, Nov 1996, pp. 374–380.

[2] X. Li, J. Le, P. Gopalakrishnan, and L. T. Pileggi, "Asymptotic probability extraction for non-normal distributions of circuit performance," in *Proc. Int. Conf. Computer-Aided Design*, Nov 2004, pp. 2–9.

[3] ——, "Asymptotic probability extraction for nonnormal performance distributions," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 1, pp. 16–37, Jan. 2007.

[4] D. Xiu, "Fast numerical methods for stochastic computations: A review," *Communications in Computational Physics*, vol. 5, no. 2-4, pp. 242–272, Feb. 2009.

[5] P. W. Tuinenga, *Spice: A Guide to Circuit Simulation and Analysis Using PSpice*, 3rd ed. Upper Saddle River, NJ: Prentice Hall PTR, 1995.

[6] `http://www.cadence.com`.

[7] `http://www.synopsys.com/Tools/Verification`.

[8] `http://www.coventor.com/products/mems/`.

[9] `http://www.coventor.com/products/coventorware/`.

[10] S.-W. Sun and P. G. Y. Tsui, "Limitation of CMOS supply-voltage scaling by MOSFET threshold-voltage variation," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 8, pp. 947–949, Aug 1995.

[11] N. Tega, H. Miki, F. Pagette, D. J. Frank, A. Ray, M. J. Rooks, W. Haensch, and K. Torii, "Increasing threshold voltage variation due to random telegraph noise in FETs as gate lengths scale to 20 nm," in *Proc. Intl. Symp. VLSI Technology*, Jun. 2009, pp. 50–51.

[12] T. Moselhy and L. Daniel, "Stochastic integral equation solver for efficient variation aware interconnect extraction," in *Proc. Design Auto. Conf.*, Jun. 2008, pp. 415–420.

[13] ——, "Stochastic dominant singular vectors method for variation-aware extraction," in *Proc. Design Auto. Conf.*, Jun. 2010, pp. 667–672.

[14] ——, "Variation-aware stochastic extraction with large parameter dimensionality: Review and comparison of state of the art intrusive and non-intrusive techniques," in *Proc. Intl. Symp. Quality Electronic Design*, Mar. 2011, pp. 14–16.

[15] T. A. El-Moselhy, "Field solver technologies for variation-aware interconnect parasitic extraction," PhD Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 2010.

[16] D. V. Ginste, D. D. Zutter, D. Deschrijver, T. Dhaene, P. Manfredi, and F. Canavero, "Stochastic modeling-based variability analysis of on-chip interconnects," *IEEE Trans. Components, Packaging and Manufacturing Technology*, vol. 2, no. 7, pp. 1182–1192, Jul. 2012.

[17] I. S. Stievano, P. Manfredi, and F. G. Canavero, "Carbon nanotube interconnects: Process variation via polynomial chaos," *IEEE Trans. Electromagnetic Compatibility*, vol. 54, no. 1, pp. 140–148, Feb. 2012.

[18] Z. Zhang, I. M. Elfadel, and L. Daniel, "Model order reduction of fully parameterized systems by recursive least square optimization," in *Proc. Intl. Conf. Computer-Aided Design*, Jun. 2011, pp. 523–530.

[19] L. Daniel, C. S. Ong, S. C. Low, K. H. Lee, and J. K. White, "A multiparameter moment-matching model-reduction approach for generating geometrically parameterized interconnect performance models," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 23, no. 5, pp. 678–693, May. 2004.

[20] V. Mehrotra, S. Nassif, D. Boning, and J. Chung, "Modeling the effects of manufacturing variation on high-speed microprocessor interconnect performance," in *Proc. Intl. Electron Devices Meeting*, Dec. 1998, pp. 767–770.

[21] K. Agarwal, D. Sylvester, D. Blaauw, F. Liu, S. Nassif, and S. Vrudhula, "Variational delay metrics for interconnect timing analysis," in *Proc. Design Auto. Conf.* New York, NY, Jun. 2004, pp. 381–384.

[22] D. S. Boning, "Variation," *IEEE Trans. Semiconductor Manufacturing*, vol. 21, no. 1, pp. 63–71, Feb 2008.

[23] L. Yu, S. Saxena, C. Hess, A. Elfadel, D. A. Antoniadis, and D. S. Boning, "Remembrance of transistors past: Compact model parameter extraction using Bayesian inference and incomplete new measurements," in *Proc. Design Automation Conf.* San Francisco, CA, Jun 2014, pp. 1–6.

[24] L. Yu, S. Saxena, C. Hess, I. M. Elfadel, D. A. Antoniadis, and D. S. Boning, "Efficient performance estimation with very small sample size via physical subspace projection and maximum a posteriori estimation," in *Proc. Design Automation and Test in Europe.* Dresden, Germany, March 2014, pp. 1–6.

[25] L. Yu, L. Wei, D. A. Antoniadis, I. M. Elfadel, and D. S. Boning, "Statistical modeling with the virtual source MOSFET model," in *Proc. Design Automation and Test in Europe.* Grenoble, France, March 2013, pp. 1454–1457.

[26] L. Yu, W.-Y. Chang, K. Zuo, J. Wang, D. Yu, and D. S. Boning, "Methodology for analysis of TSV stress induced transistor variation and circuit performance," in *Proc. Int. Symp. Quality Electronic Design.* Santa Clara, CA, March 2012, pp. 216–222.

[27] L. Yu, S. Saxena, C. Hess, A. Elfadel, D. A. Antoniadis, and D. S. Boning, "Statistical library characterization using belief propagation across multiple technology nodes," in *Proc. Design Automation and Test in Europe*, March 2015, pp. 1383–1388.

[28] L. Yu, "A study of through-silicon-via (TSV) induced transistor variation," S.M. Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 2011.

[29] C.-W. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuits and Systems*, vol. CAS-22, no. 6, pp. 504–509, Jun. 1975.

[30] S. Senturia, N. Aluru, and J. White, "Simulating the behavior of MEMS devices: Computational 3-D structures," *IEEE Computational Science and Engineering*, vol. 16, no. 10, pp. 30–43, Jan.-Mar. 1997.

[31] M. Kamon, S. Maity, D. DeReus, Z. Zhang, S. Cunningham, S. Kim, J. McKillop, A. Morris, G. Lorenzl, and L. Daniel, "New simulation and experimental methodology for analyzing pull-in and release in MEMS switches," in *Proc. IEEE Solid-State Sensors, Actuators and Microsystems Conference (TRANSDUCERS)*, Jun. 2013.

[32] G. Schröpfer, G. Lorenz, S. Rouvillois, and S. Breit, "Novel 3D modeling methods for virtual fabrication and EDA compatible design of MEMS via parametric libraries," *J. Micromech. Microeng.*, vol. 20, no. 6, pp. 064 003:1–15, Jun. 2010.

[33] Z. Zhang, M. Kamon, and L. Daniel, "Continuation-based pull-in and lift-off simulation algorithms for microelectromechanical devices," *J. Microelectromech. Syst.*, vol. 23, no. 5, pp. 1084–1093, Oct. 2014.

[34] Z. Zhang, T. A. El-Moselhy, I. M. Elfadel, and L. Daniel, "Stochastic testing method for transistor-level uncertainty quantification based on generalized polynomial chaos," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst*, vol. 32, no. 10, pp. 1533–1545, Oct 2013.

[35] Z. Zhang, I. M. Elfadel, and L. Daniel, "Uncertainty quantification for integrated circuits: Stochastic spectral methods," in *Proc. Intl. Conf. Computer-Aided Design.* San Jose, CA, Nov 2013, pp. 803–810.

[36] Z. Zhang, T. A. El-Moselhy, P. Maffezzoni, I. M. Elfadel, and L. Daniel, "Efficient uncertainty quantification for the periodic steady state of forced and autonomous circuits," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 60, no. 10, Oct 2013.

[37] Z. Zhang, X. Yang, G. Marucci, P. Maffezzoni, I. M. Elfadel, G. Karniadakis, and L. Daniel, "Stochastic testing simulator for integrated circuits and MEMS: Hierarchical and sparse techniques," in *Proc. IEEE Custom Integrated Circuits Conf.* San Jose, CA, Sept. 2014, pp. 1–8.

[38] Z. Zhang, X. Yang, I. V. Oseledets, G. E. Karniadakis, and L. Daniel, "Enabling high-dimensional hierarchical uncertainty quantification by ANOVA and tensor-train decomposition," *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 34, no. 1, pp. 63–76, Jan. 2015.

[39] Z. Zhang, T. A. El-Moselhy, I. M. Elfadel, and L. Daniel, "Calculation of generalized polynomial-chaos basis functions and Gauss quadrature rules in hierarchical uncertainty quantification," *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 33, no. 5, pp. 728–740, May 2014.

[40] D. Xiu and G. E. Karniadakis, "The Wiener-Askey polynomial chaos for stochastic differential equations," *SIAM Journal on Scientific Computing*, vol. 24, no. 2, pp. 619–644, Feb 2002.

[41] ——, "Modeling uncertainty in flow simulations via generalized polynomial chaos," *Journal of Computational Physics*, vol. 187, no. 1, pp. 137–167, May 2003.

[42] O. Le Maitre and O. Knio, *Spectral methods for uncertainty quantification: with application to computational fluid dynamics.* Springer, 2010.

[43] J. Bäck, F. Nobile, L. Tamellini, and R. Tempone, "Stochastic spectral Galerkin and collocation methods for PDEs with random coefficients: A numerical comparison," *Spectral and High Order Methods for Partial Differential Equations, Lecture Notes in Computational Science and Engineering*, vol. 76, pp. 43–62, 2011.

[44] D. Xiu, *Numerical Methods for Stochastic Computations: A Spectral Method Approach.* Princeton University Press, 2010.

[45] N. Wiener, "The homogeneous chaos," *American Journal of Mathematics*, vol. 60, no. 4, p. 897ï£¡936, Oct 1938.

[46] W. Gautschi, "On generating orthogonal polynomials," *SIAM J. Sci. Stat. Comput.*, vol. 3, no. 3, pp. 289–317, Sept. 1982.

[47] C. Soize and R. Ghanem, "Physical systems with random uncertainties: Chaos representations with arbitrary probability measure," *SIAM Journal on Scientific Computing*, vol. 26, no. 2, p. 395ï£¡410, Feb 2004.

[48] T.-W. Weng, Z. Zhang, Z. Su, Y. Marzouk, A. Melloni, and L. Daniel, "Uncertainty quantification of silicon photonic devices with correlated random parameters," *Optics Expr.*, vol. 23, no. 4, pp. 4242–4254, Feb 2015.

[49] G. H. Golub and J. H. Welsch, "Calculation of gauss quadrature rules," *Math. Comp.*, vol. 23, pp. 221–230, 1969.

[50] C. W. Clenshaw and A. R. Curtis, "A method for numerical integration on an automatic computer," *Numer. Math.*, vol. 2, pp. 197–205, 1960.

[51] L. N. Trefethen, "Is Gauss quadrature better than Clenshaw-Curtis?" *SIAM Review*, vol. 50, no. 1, pp. 67–87, Feb 2008.

[52] V. Barthelmann, E. Novak, and K. Ritter, "High dimensional polynomial interpolation on sparse grids," *Adv. Comput. Math.*, vol. 12, no. 4, pp. 273–288, Mar. 2000.

[53] T. Gerstner and M. Griebel, "Numerical integration using sparse grids," *Numer. Algor.*, vol. 18, pp. 209–232, Mar. 1998.

[54] D. Xiu and J. S. Hesthaven, "High-order collocation methods for differential equations with random inputs," *SIAM Journal on Scientific Computing*, vol. 27, no. 3, pp. 1118–1139, Mar 2005.

[55] I. Babuška, F. Nobile, and R. Tempone, "A stochastic collocation method for elliptic partial differential equations with random input data," *SIAM J. Numer. Anal.*, vol. 45, no. 3, pp. 1005–1034, Mar 2007.

[56] F. Nobile, R. Tempone, and C. G. Webster, "A sparse grid stochastic collocation method for partial differential equations with random input data," *SIAM J. Numer. Anal.*, vol. 46, no. 5, pp. 2309–2345, May 2008.

[57] ——, "An anisotropic sparse grid stochastic collocation method for partial differential equations with random input data," *SIAM J. Numer. Anal.*, vol. 46, no. 5, pp. 2411–2442, May 2008.

[58] W. J. Morokoff and R. E. Caflisch, "Quasi-Monte Carlo integration," *Journal of Computational Physics*, vol. 122, no. 2, pp. 218–230, Dec 1995.

[59] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *J. Math. Phys.*, vol. 6, pp. 39–79, 1927.

[60] J. D. Carroll and J. J. Chang, "Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, pp. 283–319, 1970.

[61] H. Kiers, "Towards a standardized notation and terminology in multiway analysis," *J. Chemometrics*, pp. 105–122, 2000.

[62] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 5, pp. 279–311, 1966.

[63] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal.*, vol. 21, pp. 1253–1278, 2000.

[64] V. De Silva and L.-H. Lim, "Tensor rank and the ill-posedness of the best low-rank approximation problem," *SIAM J. Sci. Comput.*, vol. 30, no. 5, pp. 1084–1127, 2008.

[65] I. V. Oseledets and E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use SVD in many dimensions," *SIAM J. Sci. Comput.*, vol. 31, no. 5, pp. 3744–3759, 2009.

[66] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011.

[67] I. V. Oseledets and E. Tyrtyshnikov, "TT-cross approximation for multidimensional arrays," *Linear Alg. Appl.*, vol. 432, no. 1, pp. 70–88, Jan. 2010.

[68] A. Cichoki, "Era of big data processing: A new approach via tensor networks and tensor decompositions," *arXiv Preprint, arXiv:1403.2048*, March 2014.

[69] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: Dynamic tensor analysis," in *ACM Int. Conf. Knowledge Discovery and Data Mining*, Aug. 2006, pp. 374–383.

[70] T. G. Kolda and J. Sun, "Scalable tensor decomposition for multi-aspect data mining," in *IEEE Int. Conf. Data Mining*, 2008, pp. 363–372.

[71] M. A. O. Vasilescu and D. Terzopoulos, "Multilinear analysis of image ensembles: Tensorfaces," in *Proc. Europ. Conf. Computer Vision*, 2002, pp. 447–460.

[72] D. Tao, X. Li, W. Hu, S. Maybank, and X. Wu, "Supervised tensor learning," in *Proc. Int. Conf. Data Mining*, 2005, pp. 447–460.

[73] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky, "Tensor decompositions for learning latent variable models," *arXiv Preprint, arXiv:1210.7559*, Oct 2012.

[74] A. Doostan and G. Iaccarino, "A least-square approximation of partial differential equations with high-dimensional random inputs," *J. Comp. Physcis*, vol. 228, pp. 4332–4345, 2009.

[75] A. Nouy, "Proper generalized decomposition and separated representations for the numerical solution of high dimensional stochastic problems," *Arch. Comp. Meth. Eng.*, vol. 27, no. 4, pp. 403–434, Dec 2010.

[76] A. Nouy and O. P. Le Maitre, "Generalized spectral decomposition for stochastic nonlinear problems," *J. Comp. Phys.*, vol. 228, pp. 205–235, 2009.

[77] B. N. Khoromskij and C. Schwab, "Tensor-structured Galerkin approximation of parametric and stochastic elliptic PDEs," *SIAM J. Sci. Comput*, vol. 33, no. 1, pp. 364–385, Oct 2011.

[78] V. Kazeev, M. Khammash, M. Nip, and C. Schwab, "Direct solution of the chemical master equation using quantized tensor trains," *PLOS Comp. Biology*, vol. 10, no. 3, pp. e1 003 359:1–19, March 2014.

[79] B. N. Khoromskij and I. Oseledets, "Quantics-TT collocation approximation of parameter-dependent and stochastic elliptic PDEs," *Comput. Methods in Appl. Math.*, vol. 10, no. 4, pp. 376–394, Jan 2010.

[80] S. Dolgov, B. N. Khoromskij, A. Litvnenko, and H. G. Matthies, "Computation of the response surface in the tensor train data format," *arXiv preprint, arXiv:1406.2816v1*, Jun 2014.

[81] D. Bigoni, A. P. Engsig-Karup, and Y. M. Marzouk, "Spectral tensor-train decomposition," *arXiv preprint, arXiv:1405.5713v1*, May 2014.

[82] C. Gu, E. Chiprout, and X. Li, "Efficient moment estimation with extremely small sample size via bayesian inference for analog/mixed-signal validation," in *Proc. Design Autom. Conf.* Austin, TX, Jun 2013, pp. 1–7.

[83] H. Cho, D. Venturi, and G. E. Karniadakis, "Adaptive discontinuous galerkin method for response-excitation PDF equations," *SIAM J. Sci. Comput.*, vol. 35, no. 4, pp. B890–B911, 2013.

[84] D. Zhang, *Stochastic Methods for Flow in Porous Media.* Academic Press, 2002.

[85] R. Kanj, R. Joshi, and S. Nassif, "Mixture importance sampling and its application to the analysis of SRAM designs in the presence of rare failure events," in *Proc. Design Automation Conf.* San Francisco, CA, Jun 2006, pp. 69–72.

[86] A. Singhee and R. A. Rutenbar, "Statistical blockade: Very fast statistical simulation and modeling of rare circuit events and its application to memory design," *IEEE Trans. on CAD of Integrated Circuits and systems*, vol. 28, no. 8, pp. 1176–1189, Aug. 2009.

[87] ——, "Why Quasi-Monte Carlo is better than Monte Carlo or latin hypercube sampling for statistical circuit analysis," *IEEE Trans. CAD Integrated Circuits and Systems*, vol. 29, no. 11, pp. 1763–1776, Nov. 2010.

[88] X. Li, "Finding deterministic solution from underdetermined equation: large-scale performance modeling of analog/RF circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 11, pp. 1661–1668, Nov 2011.

[89] X. Yang and G. E. Karniadakis, "Reweighted l_1 minimization method for sothcastic elliptic diffcrcntial equations," *J. Comp. Phys.*, vol. 248, no. 1, pp. 87–108, Sept. 2013.

[90] J. Tao, X. Zeng, W. Cai, Y. Su, D. Zhou, and C. Chiang, "Stochastic sparse-grid collocation algorithm (SSCA) for periodic steady-state analysis of nonlinear system with process variations," in *Porc. Asia and South Pacific Design Automation Conf.*, 2007, pp. 474–479.

[91] N. Metropolis and S. Ulam, "The Monte Carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[92] D. L. Donoho, "Compressed sensing," *IEEE Trans. Informa. Theory*, vol. 52, no. 4, pp. 578 –594, April 2006.

[93] R. Ghanem and P. Spanos, *Stochastic finite elements: a spectral approach.* Springer-Verlag, 1991.

[94] S. Vrudhula, J. M. Wang, and P. Ghanta, "Hermite polynomial based interconnect analysis in the presence of process variations," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 2001–2011, Oct. 2006.

[95] J. Wang, P. Ghanta, and S. Vrudhula, "Stochastic analysis of interconnect performance in the presence of process variations," in *Proc. Design Auto Conf.*, 2004, pp. 880–886.

[96] R. Shen, S. X.-D. Tan, J. Cui, W. Yu, Y. Cai, and G.-S. Chen, "Variational capacitance extraction and modeling based on orthogonal polynomial method," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, no. 11, pp. 1556 –1565, Nov. 2010.

[97] W. Yu, C. Hu, and W. Zhang, "Variational capacitance extraction of on-chip interconnects based on continuous surface model," in *Proc. Design Auto. Conf.*, Jun. 2009, pp. 758–763.

[98] F. Gong, H. Yu, L. Wang, and L. He, "A parallel and incremental extraction of variational capacitance with stochastic geometric moments," *IEEE Trans. VLSI*, vol. 22, no. 9, pp. 1729 –1737, Sept. 2012.

[99] H. Zhu, X. Zeng, W. Cai, J. Xue, and D. Zhou, "A sparse grid based spectral stochastic collocation method for variations-aware capacitance extraction of interconnects under nanometer process technology," in *Proc. Design Auto. Test in Europe*, Apr. 2007, pp. 1–6.

[100] I. S. Stievano, P. Manfredi, and F. G. Canavero, "Carbon nanotube interconnects: Process variation via polynomial chaos," *IEEE Trans. Electromagnetic Compatibility*, vol. 54, no. 1, pp. 140–148, Feb. 2012.

[101] ——, "Stochastic analysis of multiconductor cables and interconnects," *IEEE Trans. Electromagnetic Compatibility*, vol. 53, no. 2, pp. 501–507, May 2011.

[102] ——, "Parameters variability effects on multiconductor interconnects via hermite polynomial chaos," *IEEE Trans. Compon., Packag., Manufacut. Tech.*, vol. 1, no. 8, pp. 1234–1239, Aug. 2011.

[103] J. Fan, M. Ni, S. X.-D. Tan, Y. Cai, and X. Hong, "Statistical model order reduction for interconnect circuits considering spatial correlations," in *Proc. Design Auto. Test in Europe*, Apr. 2007, pp. 1508–1513.

[104] Y. Zou, Y. Cai, Q. Zhou, X. Hong, S. X.-D. Tan, and L. Kang, "Practical implementation of stochastic parameterized model order reduction via Hermite polynomial chaos," in *Proc. Asia South Pacific Design Auto. Conf.*, 2007, pp. 367 – 372.

[105] T. A. El-Moselhy and L. Daniel, "Variation-aware interconnect extraction using statistical moment preserving model order reduction," in *Proc. Design Automation and Test in Europe*. Dresden, Germany, March 2010, pp. 453–458.

[106] K. Strunz and Q. Su, "Stochastic formulation of SPICE-type electronic circuit simulation with polynomial chaos," *ACM Trans. Modeling and Computer Simulation*, vol. 18, no. 4, Sep. 2008.

[107] P. Manfredi, D. V. Ginste, D. De Zutter, and F. Canavero, "Stochastic modeling of nonlinear circuits via SPICE-compatible spectral equivalents," *IEEE Trans. Circuits Syst. I: Regular Papers*, 2014.

[108] R. Pulch, "Polynomial chaos expansions for analysing oscillators with uncertainties," in *Proc. MATHMOD*, 2009.

[109] ——, "Modelling and simulation of autonomous oscillators with random parameters," *Mathematics and Computers in Simulation*, vol. 81, no. 6, pp. 1128–1143, Feb 2011.

[110] ——, "Polynomial chaos for multirate partial differential algebraic equations with random parameters," *Applied Numerical Mathematics*, vol. 59, no. 10, pp. 2610–2624, Oct 2009.

[111] T.-A. Pham, E. Gad, M. S. Nakhla, and R. Achar, "Decoupled polynomial chaos and its applications to statistical analysis of high-speed interconnects," *IEEE Trans. Components, Packaging and Manufacturing Technology*, vol. 4, no. 10, pp. 1634–1647, Oct 2014.

[112] M. R. Rufuie, E. Gad, M. Nakhla, R. Achar, and M. Farhan, "Fast variability analysis of general nonlinear circuits using decoupled polynomial chaos," in *Proc. Int. Workshop on Signal and Power Integrity*, 2014, pp. 1–4.

[113] N. Agarwal and N. R. Aluru, "A stochastic Lagrangian approach for geometrical uncertainties in electrostatics," *J. Comput. Physics*, vol. 226, no. 1, pp. 156–179, Sep 2007.

[114] ——, "A domain adaptive stochastic collocation approach for analysis of MEMS under uncertainties," *J. Comput. Physics*, vol. 228, no. 20, pp. 7662–7688, Nov. 2009.

[115] ——, "A data-driven stochastic collocation approach for uncertainty quantification in MEMS," *Int. J. Numerical Methods in Eng.*, vol. 83, no. 5, pp. 575–597, July 2010.

[116] ——, "Stochastic analysis of electrostatic MEMS subjected to parameter variations," *J. Microelectromech. Syst.*, vol. 18, no. 6, pp. 1454–1468, Dec. 2009.

[117] F. A. Boloni, A. Benabou, and A. Tounzi, "Stochastic modeling of the pull-in voltage in a MEMS beam structure," *IEEE Trans. Magnetics*, vol. 47, no. 5, pp. 974–977, May 2011.

[118] R. Pulch, "Stochastic collocation and stochastic Galerkin methods for linear differential algebraic equations," available at `http://www.imacm.uni-wuppertal.de/fileadmin/imacm/preprints/2012/imacm_12_31.pdf`.

[119] A. Sandu, C. Sandu, and M. Ahmadian, "Modeling multibody systems with uncertainties. part I: Theoretical and computational aspects," *Multibody Syst. Dyn.*, vol. 15, pp. 373–395, Sept. 2006.

[120] K. Nabors and J. White, "FastCap: a multipole accelerated 3-D capacitance extraction program," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 10, no. 11, pp. 1447–1459, Nov. 1991.

[121] K. S. Kundert, *The Designer's Guide to SPICE and Spectre.* Boston, MA: Kluwer Academic Publishers, 1995.

[122] D. Calvetti and L. Reichel, "Fast inversion of Vandermonde-like matrices involving orthogonal polynomials," *BIT Numerical Mathematics*, vol. 33, no. 3, pp. 473–484, 1994.

[123] A. Narayan and D. Xiu, "Stochastic collocation with least orthogonal interpolant Leja sequences," in *SIAM Conf. Computational Science and Engineering*, Feb.-Mar. 2013.

[124] S. Weinzierl, "Introduction to Monte Carlo methods," NIKHEF, Theory Group, Amsterdam, The Netherlands, Tech. Rep. NIKHEF-00-012, 2000.

[125] "Star-HSPICE user's manual," avant! Corporation, Sunnyvale, CA, Feb. 1996.

[126] http://www.mosis.com/files/test_data/t14y_tsmc_025_level3.txt.

[127] H.-J. Bungartz and M. Griebel, "Sparse grids," *Acta Numerica*, vol. 13, pp. 147–269, 2004.

[128] K. S. Kundert, "Introduction to RF simulation and its application," *IEEE J. Solid-State Circuits*, vol. 34, no. 9, pp. 1298–1319, Sept. 1999.

[129] O. Nastov, R. Telichevesky, K. Kundert, and J. White, "Fundamentals of fast simulation algorithms for RF circuits," *IEEE Proc.*, vol. 95, no. 3, pp. 600–621, March 2007.

[130] R. Telichevesky, K. S. Kundert, and J. K. White, "Efficient steady-state analysis based on matrix-free Krylov-subspace methods," in *Proc. Design Auto. Conf.* New York, NY, Jun 1995, pp. 480–484.

[131] T. Aprille and T. Trick, "A computer algorithm to determine the steady-state response of nonlinear oscillators," *IEEE Trans. Circuit Theory*, vol. CT-19, no. 4, pp. 354–360, July 1972.

[132] P. Maffezzoni, "Efficient multi-parameter sensitivity computation of amplifier harmonic distortion," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 54, no. 3, pp. 257–261, Mar. 2007.

[133] A. K. Wilkins, B. Tidor, J. White, and P. I. Barton, "Sensitivity analysis for oscillating dynamical systems," *SIAM J. Sci. Comput.*, vol. 31, no. 4, pp. 2706–2732, April. 2009.

[134] I. Vytyaz, D. C. Lee, P. K. Hanumolu, U.-K. Moon, and K. Mayaram, "Sensitivity analysis for oscillators," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst*, vol. 27, no. 9, pp. 1521–1534, Sept. 2008.

[135] J. White and S. B. Leeb, "An envelope-following approach to switching power converter simulation," *IEEE Trans. Power Electronics*, vol. 6, no. 2, pp. 303–307, April 1991.

[136] L. Petzold, "An efficient numerical method for highly oscillatory ordinary differential equations," *SIAM J. Numerical Analysis*, vol. 18, no. 3, June 1981.

[137] A. Brambilla and P. Maffezzoni, "Envelope-following method to compute steady-state solutions of electrical circuits," *IEEE Trans Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, no. 3, pp. 407–417, March 2003.

[138] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett, "First-order incremental block-based statistical timing analysis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2170 – 2180, Oct 2006,.

[139] F. Gong, H. Yu, and L. He, "Stochastic analog circuit behavior modeling by point estimation method," in *Proc. Intl. Symp. Physical Design.* Santa Barbara, CA, March 2011, pp. 175–182.

[140] C. Michael and M. Ismail, "Statistical modeling of device mismatch for analog MOS integrated circuits," *IEEE Journal Solid-State Circuits*, vol. 27, no. 2, pp. 154–166, Feb. 1992.

[141] P. Sumant, H. Wu, A. Cangellaris, and N. R. Aluru, "Reduced-order models of finite element approximations of electromagnetic devices exhibiting statistical variability," *IEEE Trans. Antennas and Propagation*, vol. 60, no. 1, pp. 301–309, Jan. 2012.

[142] M. Frangos, Y. Marzouk, K. Willcox, and B. van Bloemen Waanders, "Surrogate and reduced-order modeling: A comparison of approaches for large-scale statistical inverse problems," *Large-Scale Inverse Problems and Quantification of Uncertainty*, pp. 123–149, 2010.

[143] "MEMS+ user's mannual," Coventor, Inc.

[144] I. M. Sobol, "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates," *Math. Comp. Sim.*, vol. 55, no. 1-3, pp. 271–280, Feb 2001.

[145] H. Rabitz and O. F. Alis, "General foundations of high-dimensional model representations," *J. Math. Chem.*, vol. 25, no. 2-3, pp. 197–233, 1999.

[146] X. Yang, M. Choi, G. Lin, and G. E. Karniadakis, "Adaptive ANOVA decomposition of stochastic incompressible and compressible flows," *J. Comp. Phys.*, vol. 231, no. 4, pp. 1587–1614, Feb 2012.

[147] M. Griebel and M. Holtz, "Dimension-wise integration of high-dimensional functions with applications to finance," *J. Complexity*, vol. 26, no. 5, pp. 455–489, Oct 2010.

[148] Z. Zhang, M. Choi, and G. E. Karniadakis, "Error estimates for the ANOVA method with polynomial choas interpolation: tensor product functions," *SIAM J. Sci. Comput.*, vol. 34, no. 2, pp. A1165–A1186, 2012.

[149] X. Ma and N. Zabaras, "An adaptive high-dimensional stochastic model representation technique for the solution of stochastic partial differential equations," *J. Compt. Phys.*, vol. 229, no. 10, pp. 3884–3915, May 2010.

[150] P. G. Constantine, E. Dow, and Q. Wang, "Active subspace methods in theory and practice: applications to kriging surfaces," *arXiv Preprint, arXiv:1304.2070v2*, Dec. 2013.

[151] J. Peng, J. Hampton, and A. Doostan, "A weighted l_1 minimization approach for sparse polynomial chaos expansion," *J. Comp. Phys.*, vol. 267, no. 1, pp. 92–111, Jun. 2014.

[152] J. Hampton and A. Doostan, "Compressive sampling of sparse polynomial chaos expansion: convergence analysis and sampling strategies," *J. Comp. Phys.*, vol. 280, no. 1, pp. 363–386, Jan. 2015.

[153] I. V. Oseledets, "TT-Toolbox 2.2," available online: `http://spring.inm.ras.ru/osel/?page_id=24`.

[154] D. R. Dereus, S. Natarajan, S. J. Cunningham, and A. S. Morris, "Tunable capacitor series/shunt design for integrated tunable wireless front end applications," in *Proc. IEEE Micro Electro Mechanical Systems*, Jan. 2011, pp. 805–808.

[155] A. K. Stamper, C. V. Jahnes, S. R. Depuis, A. Gupta, Z.-X. He, R. T. Herrin, S. E. Luce, J. Maling, D. R. Miga, W. J. Murphy, E. J. White, S. J. Cunningham, D. R. Dereus, I. Vitomirov, and A. S. Morris, "Planar MEMS RF capacitor integration," in *Proc. IEEE Solid-State Sensors, Actuators Microsyst. Conf.*, Jun. 2011, pp. 1803–1806.

[156] F. N. Fritsch and R. E. Carlson, "Monotone piecewise cubic interpolation," *SIAM J. Numerical Analysis*, vol. 17, no. 2, pp. 238 –246, April 1980.

[157] J. M. Hyman, "Accurate monotonicity preserving cubic interpolation," *SIAM J. Sci. Stat. Comput.*, vol. 4, no. 4, pp. 645 –654, Dec. 1983.

[158] R. Delbourgo and J. A. Gregory, "Shape preserving piecewise rational interpolation," *SIAM J. Sci. Stat. Comput.*, vol. 6, no. 4, pp. 967 –976, Oct. 1985.

[159] J. A. Gregory and R. Delbourgo, "Piecewise rational quadratic interpolation to monotonic data," *IMA J. Numer. Anal.*, vol. 2, no. 2, pp. 123 – 130, 1982.

[160] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, Sept 1956.

[161] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, Sept 1962.

[162] J.-N. Hwang, S.-R. Lay, and A. Lippman, "Nonparametric multivariate density estimation: a comparative study," *IEEE Trans. Signal Processing*, vol. 42, no. 10, pp. 2795–2810, Oct 1994.

[163] R. Krishnan, W. Wu, F. Gong, and L. He, "Stochastic behavioral modeling of analog/mixed-signal circuits by maximizing entropy," in *Proc. Intl. Symp. Qual. Electr. Design.* Santa Clara, CA, Mar 2013, pp. 572–579.

[164] L. Yan, L. Guo, and D. Xiu, "Stochastic collocation algorithms using l_1 minimization," *Int. J. Uncert. Quant.*, vol. 2, no. 3, pp. 279–293, Nov. 2012.

[165] I. H. Slaon and H. Wozniakowski, "When are quasi-Monte Carlo algorithms efficient for high-dimensional integrals?" *Journal of Complexity*, vol. 14, no. 1, pp. 1–33, Mar 1998.

[166] R. Pulch, "Polynomial chaos for linear differential algebraic equations with random parameters," *Int. J. Uncertainty Quantification*, vol. 1, no. 3, pp. 223–240, 2011.

[167] Z. Zhang, I. M. Elfadel, and L. Daniel, "Hierachical uncertainty quantification: Practical algorithms and numerical validation," in preparation.

[168] J. C. Wheeler, "Modified moments and gaussian quadrature," *Rochy Muntain J. Math.*, vol. 4, no. 2, pp. 287–296, Spring 1974.

[169] J. Singh and S. Sapatnekar, "Statistical timing analysis with correlated non-gaussian parameters using independent component analysis," in *Proc. Design Autom. Conf.* San Francisco, CA, Jun 2006, pp. 155–160.

[170] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 4, pp. 352 –366, April 1990.

[171] X. Wan and G. E. Karniadakis, "Long-term behavior of polynomial chaos in stochastic flow simulations," *Comput. Methods Appl. Mech. Eng.*, vol. 195, no. 41-43, pp. 5582 – 5596, Aug. 2006.