Probabilistic Interval-Valued Computation: Representing and Reasoning about Uncertainty in DSP and VLSI Design

Claire Fang Fang

April, 2005

Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
Rob A. Rutenbar, Chair
Tsuhan Chen
Larry Pileggi
Sani Nassif, IBM

Copyright © 2005 Claire Fang Fang

This research was sponsored by the Semiconductor Research Corporation and and Marco Focus Center for Circuit and System Solutions (C2S2).

Abstract

In DSP and VLSI design, there are many variational parameters that are unknown during the design stage, but significantly affect chip performance. Some uncertainties are due to manufacturing process fluctuations, others depend on the dynamic context in which the chip is used, such as input patterns, temperature and voltage. Chip designers need to consider these uncertainties as early as possible to ensure chip performance, improve yield and reduce design cost. However, it is a challenging task to model uncertainties and predict the joint impacts of them, which often either requires high computational cost or yields unsatisfactory accuracy.

Interval algebra provides a general solution to modeling and manipulating uncertainties. The idea is to replace scalar quantities with bounded intervals, and propagate intervals through arithmetic operations. A recent technique—affine arithmetic—advances the field in handling correlated intervals. However, it still produces overly conservative bounds due to the inability to consider probability information.

The goal of this dissertation is to improve the accuracy of affine arithmetic and broaden its application in DSP and VLSI design. To achieve this goal, we develop a probabilistic interval method that enhances the interval representation and computations with probability information. First, we provide a probabilistic interpretation for affine intervals based on the Central Limit Theory. Based on this interpretation, we present a probabilistic bounding method that returns less pessimistic bounds of affine intervals. Second, we propose an enhanced interval representation form that utilizes probability information to handle *asymmetric* affine intervals. This addresses a fundamental issue of current affine arithmetic, i.e., it only represents center-symmetric intervals. This restriction highly limits the accuracy of nonlinear interval functions. By introducing center-asymmetric affine intervals, we are able to design better algorithms for nonlinear interval functions. We present the improved algorithms for common nonlinear functions, with emphasis on the multiplication and the division algorithms. Finally, we also realize that in many applications, detailed probability distribution within an interval is more desirable than its bounds. Therefore, another contribution of this dissertation is to enable our interval method to estimate not only the bounds, but also the distribution within an interval. We demonstrate the effectiveness of our techniques by several applications in DSP and VLSI design.

To my parents, husband and family.

Acknowledgments

I am extremely grateful to Prof. Rob A. Rutenbar, my thesis advisor, for giving me the right amount of freedom and guidance during my thesis research. In the early stages, he gave me tremendous support and encouragement when I decided to work on several different projects to search for concrete thesis ideas. That experience helped me identify my strength and weakness as a researcher and find the focus of this thesis. Rob has been a great source of inspiration for me, with his profound insight of the field, endless passion for research and unique sense of humor. He not only taught me how to be a better researcher, but also set a great example for how to effectively communicate research ideas, which is extremely important to all the achievements I have today. I also thank him for giving me the many opportunities to present my research to broad audiences, which helped me become more confident and mature, in work, and in life.

I am deeply thankful to Prof. Tsuhan Chen, my co-advisor, for standing behind me the entire way. He has given me much more than research guidance. When I first came to CMU and felt uncertain and insecure about the future, he used his own experience to show me whom I can be. His annual how-to-succeed-in-the-AMP-lab advices guided my way throughout my graduate studies. I also benefited enormously from his positive thinking and meticulous attention to detail and quality. Every time I finished discussion with him, I felt nothing but cheerfulness and were ready to march toward the next milestone. His engaging arguments and detailed feedback contributed greatly to my thesis research. I am very grateful to have had him as my co-advisor.

Many thanks to my thesis committee members Dr. Sani Nassif and Prof. Larry Pileggi for their feedback and suggestions that helped to improve the overall quality of this dissertation. Special thanks to Prof. James Hoe and Dr. Markus Püschel for letting me be part of the SPIRAL project. I learned many valuable research skills from them.

I am grateful to all the colleagues and friends with whom I spent my time as a graduate student at Carnegie Mellon University. Thanks to my group-mates Hongzhou Liu, Hui Xu, Zhong Xiu, Edward Lin, James Ma, Saurabh Kumar Tiwary, Dong Chen and Smiriti Gupta and Amith Singhee. They have contributed to my research through many fruitful discussions and hands-on help. Thanks to all the members in the AMP (Advanced-Multimedia-Processing) lab, Deepak Turaga, Trista Chen, Howard Leung, Xiaoming Liu, Cha Zhang, Wende Zhang, Sam Chen, Jessie Hsu, Jack Yu and Kate Shim. We not only exchange research ideas, but also shared pleasant memories through many fun group activities. Special thanks to my friends Shuheng Zhou, Yang Xu, Xin

Li, Peng Li, Xue Bai, Cuihong Li, Jessica Guo and Vivien Xiong for sharing my highs and lows and enriching my stay here at CMU. There are many other friends who have always supported me. I can not possibly list all their names here. I would like to thank them all.

I am very grateful to Lyz Knight and Roxann Martin for all of the time and work they contributed to making my life as a graduate student much easier. I also would like to thank Lynn Phillibin and Elaine Lawrence from the graduate office who continually provided guidance.

I would like to express my earnest gratitude to my family for their love and support, without which none of my achievements would have been possible. Over the many years, my father, Baoxin Fang, has been constantly reminding me the importance of hard work and perseverance, and my mather, Yun Dang, always encouraged me to pursue my dreams even if she had to sacrifice a lot. I feel deeply indebted to them, especially when I could not be around spending Chinese holidays with them. I am grateful to my Aunt, Ping Dang, and cousin, Yuan Jiang, for having great online chats with me on weekends. They shared all the happiness and troubles in my life, helped me decompress, and more importantly, made me feel closer to home. Thanks to my parents-in-law, Jing Ke and Tong Liang, for their love and trust in me. Finally, and most importantly, I deeply thank my husband, Yan Ke, for the love and unconditional support he has given me in the past three years. He has made me a more joyous person and enriched my life beyond measure. I am so thankful to have such a wonderful family. With deepest appreciation, I dedicate my work and this dissertation to them.

This research was sponsored by the Semiconductor Research Corporation and and Marco Focus Center for Circuit and System Solutions (C2S2).

Table of Contents

1 Introduction **1**

1.1 Motivation 1

1.1.1 Design under uncertainty 1

1.1.2 Explicit PDF techniques 3

1.1.3 Interval-valued techniques 4

1.1.4 Middle ground 5

1.2 Main contributions of the dissertation 5

1.3 Organization of the dissertation 7

2 Background **9**

2.1 Interval Arithmetic 9

2.1.1 Introduction 9

2.1.2 Computing with interval arithmetic 10

2.1.3 Limitations of interval arithmetic 11

2.2 Affine Arithmetic 12

2.2.1 Introduction 12

2.2.2 Computing with affine arithmetic 13

2.2.3 Limitations of affine arithmetic 15

3 Probabilistic Bounding for Affine Intervals **17**

3.1 Motivation 17

3.2 Core Theories 19

3.2.1 Affine interval and Gaussian distribution 19
3.2.2 Probabilistic bounds for affine intervals 21
3.2.3 Initialization of affine intervals from given probabilistic information 28
3.3 Application—Finite Precision Analysis for DSP Design 29
3.3.1 Background 30
3.3.2 Solution Outline 41
3.3.3 Fixed-Point Range and Error Analysis via Affine Arithmetic 43
3.3.4 Floating-Point Range and Error Analysis via Affine Arithmetic 51
3.3.5 Experimental Results 54
3.3.6 Demonstration Applications 60
3.4 Summary 66

4 Asymmetric Probabilistic Bounding for Interval Operations 69
4.1 Motivation 70
4.1.1 The overshoot problem in unary non-affine functions 70
4.1.2 The pessimism in multiplication 71
4.1.3 The pessimism in division 72
4.2 Core Theories 74
4.2.1 Enforcing asymmetric bounds 74
4.2.2 Unary operations on an asymmetric affine interval 77
4.2.3 Binary operations on asymmetric affine intervals 80
4.2.4 Multiplication on asymmetric affine intervals 86
4.2.5 Division on asymmetric affine intervals 96
4.3 Experimental Results 104
4.3.1 A multiplication chain 106
4.3.2 Cholesky Decomposition 110
4.4 Summary 119

5 Analyzing the Probability Distribution within an Asymmetric Affine Interval 121
5.1 Motivation 121

5.2 Key Techniques . . . 126
5.2.1 Representing input distributions with intervals . . . 126
5.2.2 Center adjustment . . . 128
5.2.3 Asymmetric PDF generation . . . 128
5.2.4 PDF curve smoothing . . . 132
5.3 Experimental Results—Distribution Analysis . . . 135
5.3.1 Background: the soft-max approximation . . . 138
5.3.2 A single soft-max operator . . . 140
5.3.3 Binary tree with soft-max operators . . . 144
5.3.4 Viterbi trellis with soft-max operators . . . 150
5.4 Summary . . . 152

6 Conclusions and Future Work **155**
6.1 Contributions . . . 155
6.2 Future Work . . . 157
6.3 Closing Remarks . . . 159

A Proof of Equation (3.4) **161**

B Solution to Equation (4.17) **163**

C Parameters in Equation (4.21) **165**

D Distribution of the ratio between two correlated normal random variables **167**

E Modeling Gate Delay with Soft-Max **171**
E.1 Motivation: Underestimation by the max-delay model . . . 171
E.2 Soft-max modeling for NAND gates . . . 173
E.3 Soft-max modeling for NOR gates . . . 174
E.3.1 Summary . . . 176

List of Figures

2.1 Minimax approximation for unary non-affine functions 14

3.1 Distribution inside an affine interval 20
3.2 Hard bounds and probabilistic bounds 22
3.3 Polygon construction from a 2D affine interval 23
3.4 Approximating a polygon by an ellipse 25
3.5 Distribution inside a 2D affine interval 26
3.6 Polygon and confidence ellipses 27
3.7 Finite-precision effects on decoded video 33
3.8 Video quality vs. fraction width 37
3.9 An example of error cancellation 40
3.10 AA-based computation model 43
3.11 Example: comparison of AA- and IA-based error analysis 44
3.12 Data-flow of the IDCT algorithm 57
3.13 Probabilistic bounds for WHT64 57
3.14 Accuracy of error and range analysis 59
3.15 Correlation of pixels 61
3.16 DCT design procedure 62
3.17 Error bound vs. mantissa width in DCT design 63
3.18 An example of feedback systems—a second order IIR filter 64
3.19 Convergence of error estimation in the IIR filter design 65
3.20 Error bound vs. Fraction width in IIR filter design 65

3.21 Comparison of four DCT algorithms 67

4.1 The exp function on affine interval 71
4.2 Overestimation in multiplication 73
4.3 Histogram of x/y 74
4.4 Representing asymmetric region with asymmetric affine intervals 75
4.5 Abstract algorithm for unary functions with asymmetric bounding 78
4.6 Performing the exp function on an affine interval 78
4.7 The $\exp$ function on an asymmetric affine interval 79
4.8 The $\log$ function on an asymmetric affine interval 80
4.9 The reciprocal function on an asymmetric affine interval 81
4.10 The sqrt function on an asymmetric affine interval 82
4.11 Approximation criteria for non-affine functions 84
4.12 Examples of U^{xy} 89
4.13 An example of the new multiplication algorithm 91
4.14 Ellipse tracing to find the extremes of uv 95
4.15 The improved algorithm for the multiplication on asymmetric affine intervals . . . 97
4.16 Division by the minivolume approximation 101
4.17 Tracing the bounding box of an ellipse 103
4.18 The enforced bounds for $z = x/y$ 104
4.19 The improved algorithm for the division on asymmetric affine intervals 105
4.20 Experimental results on a multiplication chain 108
4.21 Comparison of accuracy (tightness ratio) with the original AA 115
4.22 Accuracy measurements for Experiment A 116
4.23 Accuracy measurements for Experiment B 117
4.24 Accuracy measurements for Experiment C 118

5.1 The distributions produced from interval analysis for the non-affine functions . . . 125
5.2 The distributions produced from interval analysis after center adjustment 129
5.3 The distributions produced from interval analysis by utilizing asymmetric bounds . 131

5.4 Kernel functions 133
5.5 Effects of different smoothing kernels on the PDF of the $1/x$ function 134
5.6 Effects of different smoothing bandwidths on the PDF of the $\log(x)$ function . . . 134
5.7 The distributions produced from interval analysis after curve smoothing 136
5.8 Flow chart of PDF estimation 137
5.9 Soft-max vs. max 139
5.10 Distribution analysis on a single soft-max operator 141
5.11 Performance on a soft-max operator with larger k 143
5.12 Distribution analysis on a single soft-max function under four different variation settings 145
5.13 A binary tree of the soft-max operators 147
5.14 Distribution analysis on a soft-max binary tree (input variation = 18%) 148
5.15 Distribution analysis on a soft-max binary tree (input variation = 72%) 148
5.16 A Hidden Markov Model example 151
5.17 The structure of a Viterbi trellis with the soft-max operators 151
5.18 Distribution analysis on a soft-max Viterbi trellis 152

A.1 Coordinate transformation 162

E.1 Simulation result compared to the max-delay model (NAND gate) 172
E.2 Soft-max modeling results for a NAND gate 173
E.3 The model parameter k vs. input slopes 174
E.4 Simulation result compared to the max-delay model (NOR gate) 175
E.5 Soft-max modeling results for a NOR gate 176

List of Tables

3.1 Pessimistic bounds of the affine intervals 18
3.2 Examples of format propagation rules 35
3.3 Examples of error propagation rules 39
3.4 Comparison between AA-based and IA-based error analysis 56
3.5 Comparison of CPU time (fixed-point) 60
3.6 Comparison of CPU time (floating-point) 60
3.7 Constant growth of uncertainty terms 64

4.1 The four roots of equation (4.17) 94
4.2 Accuracy comparison for more iterations 109
4.3 Accuracy in the case of a large number noise terms 110
4.4 Summary of Experiment A, B, and C 114
4.5 Accuracy of Cholesky decomposition on sparse matrices of size 1000×1000 . . . 114

5.1 Estimation error for the distribution analysis on a soft-max operator (k = 0.05) . . . 142
5.2 Estimation error for the distribution analysis on a soft-max operator (k = 1) 143
5.3 Four different variation settings 143
5.4 Estimation error comparison for four different variation settings 146
5.5 Estimation accuracy (relative errors of percentiles) for binary trees 147
5.6 Computational cost of the distribution analysis on a soft-max binary tree 149
5.7 Rough speedup comparison of distribution analysis methods 149
5.8 Estimation accuracy for Viterbi trellis 153

E.1 Underestimation vs. input slope ($C_l = 0.02pF$) 172

Chapter 1

Introduction

We often need to perform engineering analysis in situations where the parameters are not precisely known or variable in different contexts. Interval-valued analysis [66] is a historically important method to model and analyze uncertainty. This dissertation develops a novel probabilistic approach to interval computations: by adding probabilistic information to intervals, we improve the accuracy of interval computations, and hence broaden its application in DSP and VLSI design.

In this chapter, we first introduce the motivation of this thesis, from both the application side and the technique side. Next, we present the main contributions of our work, detailing the improvements offered by the new techniques. Finally, we give the organization of the remaining part of this dissertation.

1.1 Motivation

1.1.1 Design under uncertainty

In DSP and VLSI design, there are many variational parameters that are unknown during the design stage, but significantly affect chip performance (such as correct functionality, speed, and power consumption). Some uncertainties are due to manufacturing process fluctuations, others depend on the dynamic context in which the chip is used, such as input data patterns, operating temperature and voltage. Chip designers need to consider these uncertainties as early as possible to ensure chip

performance, manage yield and reduce design cost. However, it is a challenging task to model such uncertainties and predict the joint impacts of them, which often either requires high computational cost or yields unsatisfactory accuracy.

One example is error analysis in finite-precision DSP design. Modern DSP applications are typically prototyped using floating-point arithmetic for large dynamic range and high precision, but often implemented with finite custom precision arithmetic to reduce hardware cost and power consumption. To determine the data format (in particular, the bitwidth) of the arithmetic unit, an inevitable task is to estimate the error caused by limited precision and verify whether the functional performance meets certain standards. However, finite-precision error is highly dependent on the application of the DSP chip and its input data. Therefore, it calls for an efficient error analysis approach that accurately models the uncertainties and estimates the maximum error. This not only will provide valuable guidance to DSP designers, but also can be integrated into high-level synthesis tools to help design automation.

Another recently popular example is timing analysis in VLSI design, which tries to predict the operating speed of a chip before it is manufactured. There are two types of uncertainties in timing analysis. *Static* uncertainties refer to those caused by uncontrollable variations in manufacturing process (e.g. gate length, dopant concentrations and oxide thickness). *Dynamic* uncertainties are those due to environmental variations (e.g. voltage, temperature and crosstalk). All these variations lead to uncertainties in device behaviors, and hence add randomness to the key circuit-level properties, such as speed and power. More importantly, these variations have been continuously increasing, due to the constant scaling of CMOS technology [69]. Moreover, they exhibit complicated spatial correlations, making timing analysis a very challenging task [2]. Conventional static timing analysis (STA), which analyzes the worst case timing behavior, tends to provide pessimistic results and leads to over design. To overcome this problem, there has been intensive research on statistical static timing analysis (SSTA) in recent years, which proposes to analyze a probability distribution for the circuit delay, taking into account the parameter variations and their correlations [1,13,23,50,87,88].

The most commonly used method for dealing with uncertainty is Monte Carlo simulation. First, a distribution model is built for each variational parameter. Then, it randomly samples the parameters according to their distributions and simulates the design to obtain the performance measure

(e.g, the finite-precision error or the circuit delay). While this method is effective, it often requires a large number of test runs to accurately capture the performance, thus making it computationally expensive. Further, it provides little insight on how the design choices affect the circuit performance and where these can be potentially improved. Consequently, each time a design choice is significantly changed, it requires a complete simulation of the design, which makes it prohibitively costly to be included in automatic design optimization.

Alternatively, researchers have been actively seeking analytical approaches to handle uncertainties. They can be divided into two categories: one that models uncertainties with some explicitly "fit" *PDFs* (Probability density function), and the other models uncertainties with *intervals*.

1.1.2 Explicit PDF techniques

One option for describing uncertain information is to render each variable in the form of some explicitly known, analytically tractable PDF. Typically, the variational parameters are assumed to follow normal (Gaussian) distributions, and their correlations are specified by a correlation matrix. Statistical Static Timing Analysis (SSTA) is perhaps the most successful example of the industrial applications of this strategy to date [13, 23, 50, 54, 88]. The idea is to propagate normal distributions through each atomic operation and obtain a normal distribution for the target result. For SSTA, this means pushing correlated normal distributions representing signal arrival times through addition, subtraction, maximum, and minimum operators. Luckily, this works well in practice in the SSTA application. Linear operators (add, subtract) preserve normality. In the empirically important case of normal-valued circuit delays with variations that are not too dissimilar, there are classical normal approximations to the minimum and maximum of a pair of correlated normals [15] that work well. Unfortunately, the nonlinearities of max/min are not, in the general case, well modeled by normals: distributions with very different variances yield heavy-tailed PDFs in such cases. This is an unfortunately common problem, for example, representing a product or a quotient of correlated normals is not straightforward [61, 63].

Strategies that rely on explicit PDFs appear to us to be application specific. That is, where they work, they work very well, such as in the SSTA task. These techniques can be extremely accurate and efficient. However, we are left with the problem that in some tasks, we have no explicit

algorithms for "pushing" the PDFs through the operations we need to model statistically.

1.1.3 Interval-valued techniques

An alternative strategy for representing uncertainties, and one with a much longer historical development, is the use of interval-valued techniques. Originally introduce by Moore in 1966 [66], the idea has been widely used, analyzed, and extended. In contrast to methods based on PDFs, intervals—at least in classical form—model the range of an uncertainty, but not its explicit probability distribution. This is obviously much more efficient, but concomitantly, also less accurate.

The earliest interval technique is *interval arithmetic* (IA) [66], in which an uncertain quantity is specified by an upper bound and a lower bound. For each arithmetic operation in the analysis, the goal is to compute a tight bound for the output, which is a much easier task than computing its distribution. By quickly propagating intervals through the analysis, one can obtain a bound for the final analysis output. IA has been applied to various problems in VLSI design, such as switch-level simulation [36], RC timing analysis [37] and even placement [36]. The main attraction of IA is speed. Usually the run time of an interval algorithm is greater than that of its non-interval counterpart by only a small constant factor.

However, a major drawback with IA is that it often provides overly conservative bounds due to the lack of correlation handling. This issue highly limits the accuracy of classical interval computations, as shown in [36]. Consider the extreme example of $x - x$. In IA, when the quantity x is random and represented by an interval $[x]$, the corresponding interval computation $[x] - [x]$ does not result in the interval [0, 0], since it assumes the first operand and the second operand are independent, while in fact, they represent the same uncertain quantity.

A relatively new and more sophisticated interval-valued technique is *affine arithmetic* (AA) [17]. It models uncertainty with a first-order polynomial form:

$$\widehat{x} = x_0 + x_1\varepsilon_1 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n, \quad \text{with} -1 \leqslant \varepsilon_i \leqslant 1.$$

It offers higher accuracy than IA, because the interval form includes partial dependency information, and hence correlations in interval computations are efficiently handled. Applications of AA in VLSI design have included analog circuit sizing [51], and circuit tolerance analysis [28].

However, affine arithmetic has its limitations too. First, due to the special form of its interval representation, it has significantly more storage and computational requirements than IA. Second, in order to preserve the consistent interval representation, it introduces many approximations in nonlinear arithmetic operations, resulting in very pessimistic results. Third, for the final analysis result, it returns bounds without any indication of the probability of occurrence. If the probabilities of these bounds are extremely small, then the estimation is again overly conservative.

1.1.4 Middle ground

The explicit PDF approaches provide more information about the uncertainties, but suffer from difficult distribution propagation and correlation handling; on the other hand, the interval approaches (especially affine arithmetic) are simple to implement and efficiently handles correlations, but may be too conservative due to the lack of probability information. In this dissertation, we show that middle ground between these two approaches is an attractive place to look for tractable compromises that mitigate some of the problems with each approach. Although the two approaches are based on completely different representations, we show that a novel combination of the two can take advantage of both of the approaches.

1.2 Main contributions of the dissertation

In order to improve the accuracy of interval-based analysis and hence broaden its application in DSP and VLSI design, we develop a probabilistic enhancement to a recent interval technique—affine arithmetic. The main contributions of this dissertation include the following:

- **Associate an affine interval with a probability distribution**. One drawback with the existing affine arithmetic is that the bounds it estimates are often too pessimistic, with extremely low likelihood. This problem is severe for large scale problems. Interestingly, based on the special representation form of an affine interval, we discover that in common circumstances, an affine interval often implies a normal distribution. Associating a distribution with an interval is conceptually a jump from the traditional interval techniques. This probabilistic interpretation reduces the pessimism of affine arithmetic in the following two ways. First, we are

able improve the accuracy of interval computations by utilizing the probability information associated with the input affine intervals. More specifically, for a function $z = f(x, y)$, we are able to identify where in the joint range of x and y the probability is extremely low, and accordingly, exclude those areas when computing the output interval. Second, when we estimate the final bounds at the end of interval analysis, instead of blindly returning the extreme limits, we return a tighter confidence interval based on the probability distribution.

- **Enable AA to handle non-center-symmetric intervals**. According to its representation form, an affine interval defines a center and an interval, and the latter has to be symmetric around the center. However, interval computations do not always yield a symmetric interval, even when the inputs are symmetric intervals. Therefore, in existing AA, many interval computations utilize conservative approximations, in order to preserve center symmetry. Sometimes, such sacrifice leads to even more conservative results than IA, completely demolishing the advantage of AA. This issue highly limits the application of AA in many nonlinear problems. In this dissertation, we provide a simple, but powerful, enhancement that enables non-center-symmetric intervals. We also develop new interval computation algorithms to handle asymmetric affine intervals.

- **Improve the accuracy of nonlinear interval functions**. Existing algorithms for nonlinear interval computations in AA are very pessimistic. Our improvements of these functions are twofold. As we have stated, enabling asymmetry of affine intervals certainly reduces conservative approximations. The other important source of improvements is a new interval computation rule, called the *minivolume approximation*. It significantly tightens the output interval for multiplication and division, and also provides a general guideline for other binary functions.

- **Provide means to estimate asymmetric probability distribution through interval computations**. Traditionally, interval techniques have been used to provide a bound estimation. However, in many problems, the detailed probability distribution within the bounded interval is more desirable. For example, in statistical static timing analysis, the goal is to estimate a PDF for the circuit delay. On the other hand, probabilistic approaches usually do not handle asymmetric distributions (common in nonlinear applications) very well. In this disserta-

tion, we develop techniques to estimate the probability distribution, which could be center-asymmetric, within an affine interval. To distinguish from other probabilistic approaches, our interval-based distribution analysis propagates *intervals*, to be more precise, *asymmetric* affine intervals, as opposed to probability distributions. Therefore, it still enjoys the attractive correlation handling of affine arithmetic. At the end of interval analysis, the probability distribution is estimated from the output asymmetric interval.

1.3 Organization of the dissertation

The rest of the dissertation is organized as follows. Chapter 2 provides background on relevant interval techniques, in particular, interval arithmetic and affine arithmetic. We first give a brief introduction of their computation rules and applications, and then discuss their limitations.

Chapter 3 describes a probabilistic bounding method that significantly reduces pessimism in affine arithmetic. We begin the chapter by developing an inherent connection between affine intervals and probability distributions, and then propose a new probabilistic bounding method for 1D and 2D affine intervals. In the rest of this chapter, we apply AA with the new bounding method to a problem in finite-precision DSP design. Through experiments on several common DSP kernels, we demonstrate that the AA-based approach achieves high accuracy that is comparable to Monte Carlo simulation, but with four to five orders of magnitude speedup.

Chapter 4 introduces a method to handle asymmetric affine intervals, focusing on its impact on nonlinear interval functions. We first motivate our work by analyzing the pessimism in the existing algorithms of nonlinear interval functions. Then, we discuss the key techniques and the proposed algorithms for common nonlinear functions. In addition, during the discussion of binary functions, we propose a novel minivolume approximation rule which further improves the accuracy for multiplication and division. Finally, we demonstrate the improvements by applying the new technique to a DSP algorithm—the Cholesky decomposition.

Chapter 5 presents how we analyze the probability distribution within an asymmetric affine interval. The first half of the chapter is dedicated to the three key techniques that are essential to distribution analysis. In the second half, we test the accuracy and applicability of the proposed

technique on nonlinear applications that involve a classical continuous approximation to the highly nonlinear maximum operator, called the *soft-max*. Soft-max is an attractive test case since it highlights in one function the worst problems of common arithmetic (not only addition and subtraction, but also multiplication and division) and transcendental (exponential and logarithmic) functions.

Finally, Chapter 6 summarizes the conclusions of the dissertation, discusses the limitations of our work, and ends with directions for future work.

Chapter 2

Background

Interval techniques have been in existence almost four decades. The first technique in the field, *interval arithmetic*, was published in 1966 [66]. Since then, interval analysis has become an intensively studied branch of computational mathematics. A recent refinement, *affine arithmetic*, was proposed in 1993 [17], aiming for better correlation handling than interval arithmetic. In this chapter, we provide relevant background on these two techniques. We will cover the basic rules for interval computations, the advantages and the limitations of each technique.

2.1 Interval Arithmetic

2.1.1 Introduction

Interval arithmetic (IA) was first introduced by R. E. Moore as a range-based model for numerical computations with limited-precision floating-point numbers. In IA, each quantity is represented by an interval

$$[x] = [\underline{x}, \overline{x}]$$

which bounds all possible values of x. Computations on real numbers are therefore replaced by computations on intervals. The fundamental theorem of interval arithmetic states that the computed interval should always contain the possible values of the corresponding quantity.

The early attraction of interval arithmetic is the ability to uncover the uncertainties in machine

computations with limited precisions. Each floating-point number is associated with an interval that bounds all possible real values represented with this floating-point number. The width of the computed interval indicates the "safeness" of the machine computation. Later on, interval arithmetic was developed to attack many fundamental problems, including linear equations [46, 70], nonlinear equations [79, 80], and global optimization [35, 78]. It is usually faster than non-interval methods (for example, Monte Carlo simulation and generic algorithms) for finding all solutions or global optima. In the recent decade, due to the development of public software packages [8, 39, 74], IA has been widely adopted in many different areas, such as computer graphics, chemical engineering, electrical engineering, manufacturing quality control, economics, etc.

2.1.2 Computing with interval arithmetic

For every function $f(x, y, ..)$ on real numbers, there is a corresponding interval extension, mapping from intervals to intervals. The result should be the smallest interval that contains all possible values that are potentially reached by any combination of the input values, assuming x, y, ... vary independently over the given ranges.

For elementary functions, such as $+$, $-$, $\times$, $\div$, their interval extensions are fairly straightforward. We only need to calculate the results for any combination of the endpoints of the inputs. Then from these different cases, the maximum and the minimum values can be identified, which forms the bounds for the resulting interval. For example,

$$\begin{aligned}
[x] + [y] &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \\
[x] - [y] &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}], \\
[x] \times [y] &= [\min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}], \\
\frac{[x]}{[y]} &= [\min\{\underline{x}/\underline{y}, \overline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\overline{y}\}, \max\{\underline{x}/\underline{y}, \overline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\overline{y}\}].
\end{aligned} \tag{2.1}$$

Note that for division, the interval extension is valid only when $[y]$ does not include zero. In these four elementary functions, the maxima and the minima are exact. However, there are cases where determining the exact maxima and minima is *not* easy. It is thus acceptable to have computable, tractable bounds that are conservative *estimates* of the true bounds.

For complicated functions that are composed of elementary functions, their interval extensions

are achieved by concatenating the corresponding interval extensions of the elementary functions in the same computational order.

2.1.3 Limitations of interval arithmetic

A major drawback with IA is the inability to express the *correlations* among intervals: all the intervals are assumed to be independent, even if their corresponding quantities are dependent. This property highly limits the accuracy of interval computations. If the inputs of a function $f(x, y)$ are correlated through prior computations or some other constraints, there maybe certain unreachable regions in the joint range $[x]\times[y]$. The resulting interval computed based on the entire joint range may be overly conservative compared to the true range of the output quantity.

For example, in IA, the interval subtraction $[x] - [x]$ does not result in the interval [0, 0], since it assumes the first operand and the second operand are independent, while in fact, they represent the same quantity. Similarly, the function $[x]/[x]$ does not lead to the interval [1, 1]. These are extreme examples. In general, for addition and multiplication, if the inputs have negative correlations, the resulting interval tends to be wider than the true range, and for subtraction and division, positive input correlations usually lead to a conservative estimate.

The over-conservatism of IA is especially serious in a long computation chain, where the results computed at one stage are the inputs to the subsequent stages. As more computations are evaluated down the chain, the accuracy of the computed intervals rapidly decreases, and the intervals soon become uselessly wide. This problem, often referred to as “error explosion”, hinders the application of interval arithmetic.

Various techniques have been proposed to alleviate the error explosion problem. They share a common strategy, i.e., to reduce unfavorable correlations. Sometimes this can be achieved by rearranging the computations so that the occurrences of a variable in a computation chain are minimized. Another technique is to have a single “macro interval operation” for several steps of elementary interval operations. A trivial but important example is the evaluation of powers $f = x^n$. If the interval extension is implemented by a sequence of interval multiplications, the resulting interval is too wide, as the correlation is ignored. In contrast, it can be implemented as a special routine and returns the accurate interval. Nevertheless, the power of these remedies is very limited. For large applications

where thousands of interval operations are involved, the error explosion is still a severe problem. To find more advanced details of interval arithmetic, please refer to [3, 67, 77].

2.2 Affine Arithmetic

2.2.1 Introduction

Affine arithmetic (AA) was proposed in [17] to address the "error explosion" problem in the conventional IA. In AA, each quantity x is represented by an *affine form* $\widehat{x}$, which is a first-degree polynomial:

$$\widehat{x} = x_0 + x_1\varepsilon_1 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n, \quad \text{with} -1 \leqslant \varepsilon_i \leqslant 1.$$

This representation not only contains the information on the bounds, but also reveals the underlying uncertainty components. The first term, x_0, is called the *central value*, and the bounds of $\hat{x}$ (equal to $x_0 - \sum_i^n |x_i|$ and $x_0 + \sum_i^n |x_i|$) are symmetric to this central value. Each ε_i, called a *noise symbol*, stands for an independent component of the total uncertainty and lies in the interval [-1, 1]; the corresponding coefficient x_i gives the magnitude of that component.

The key appealing feature of AA is that one noise symbol may contribute to the uncertainties of two or more variables, indicating correlations among them. The corresponding coefficients determine the magnitude and the sign of the correlation. When correlated intervals are involved in a computation (such as $[x] - [x]$), the shared uncertainty terms may cancel out, leading to a tighter output interval. Further, the correlations between the output of a computation and its inputs are also expressed through noise symbol sharing. It is the sharing of noise symbols that keeps track of the correlations occur in an application and contributes to more accurate interval computations. This advantage is especially noticeable in computations that are highly correlated or of great computational depth.

Affine arithmetic has been successfully applied to global optimization [21, 65], computer graphics [17, 20, 22, 81], computer vision [31, 32], and analog circuit sizing [51]. In [51], affine arithmetic is applied to branch-and-bound optimization in the continuously-valued sizing problem and to calculating guaranteed bounds on the true worst-case performance range.

2.2.2 Computing with affine arithmetic

For every function on real numbers, there is a corresponding interval computation that takes inputs in affine forms and returns an affine form for the output. Same as IA, this output interval should contain all values that are possibly reached.

If the function is an affine function of its arguments, then its affine interval extension is straightforward: a direct symbolic operation on the input affine forms leads to an affine form. In particular, let us consider the affine functions $x \pm y$, αx, and $x \pm \zeta$, for any $\alpha, \zeta \in \mathbf{R}$. Suppose the input affine forms are

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots x_n\varepsilon_n$$
$$\hat{y} = y_0 + y_1\varepsilon_1 + y_2\varepsilon_2 + \cdots y_n\varepsilon_n.$$

The corresponding interval extensions for these affine functions are

$$\begin{aligned} \hat{x} \pm \hat{y} &= (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n \\ \alpha\hat{x} &= (\alpha x_0) + (\alpha x_1)\varepsilon_1 + \cdots + (\alpha x_n)\varepsilon_n \\ \hat{x} \pm \zeta &= (x_0 \pm \zeta) + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n. \end{aligned} \tag{2.2}$$

For non-affine functions (e.g., square root, multiplication), the result from direct symbolic operation is not exactly an affine form. Consider a non-affine function $f(x, y)$. We first seek an affine function $f^*(x, y) = Ax + By + C$ to approximate f, and then append an extra term $D\varepsilon_k$ to represent the error introduced by this approximation. Here, ε_k is a new noise symbol, independent of any other noise symbols in the computation, and D is an upper bound for the approximation error.

A general rule for selecting the approximating affine function is the *Chebyshev (or minimax) approximation* rule, i.e., to minimize the maximum absolute error between f and f^* over the joint range of the function inputs. The corresponding algorithms for unary functions are introduced in [17]. Here, we offer a brief description of the minimax affine approximation for a unary function $z = f(x)$, illustrated by the $exp(x)$ function in Figure 2.1. Let $f^*(x) = Ax + B$ be its affine approximation, and the interval $\hat{x}$ is bounded by $[a, b]$. Then,

- The coefficient A is simply $(f(b) - f(a))/(b - a)$, the slope of the line $r(x)$ that interpolates the points $(a, f(a))$ and $(b, f(b))$.

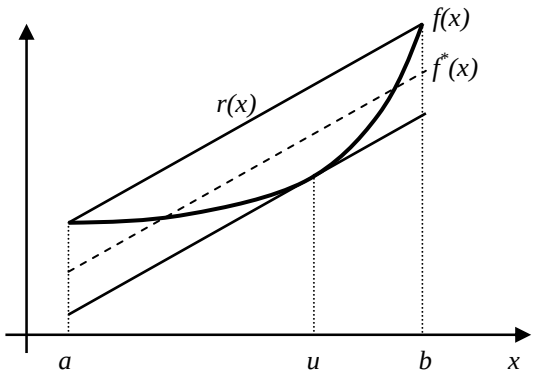


Figure 2.1: Minimax approximation for unary non-affine functions

- The maximum absolute error occurs twice (with the same sign) at the endpoints a and b, and once at an internal point u where $f'(u) = A$.
- The term C equals to $(f(u) + r(u))/2 - Au$, and the approximation error is bounded by $D = |f(u) - r(u)|/2$.

In the figure, we can see that the discrepancy between $f(x)$ and $f^*(x)$ is in fact a function of the value of x. It introduces loss of correlation information when we use an independent uncertainty term $D\varepsilon_k$ to represent this discrepancy.

For binary functions, the Chebyshev approximation is not established in [17]. Instead, they use a simple, but very conservative, approximation for the output affine form. In particular, for the multiplication $z = xy$, where the inputs are $\hat{x} = x_0 + \sum_{i=1}^{N} x_i\varepsilon_i$ and $\hat{y} = y_0 + \sum_{i=1}^{N} y_i\varepsilon_i$, a direct multiplication of the inputs results in

$$\hat{x} \cdot \hat{y} = x_0 y_0 + \sum_{i=1}^{N} (y_0 \cdot x_i + x_0 \cdot y_i)\varepsilon_i + (\sum_{i=1}^{N} x_i\varepsilon_i)(\sum_{i=1}^{N} y_i\varepsilon_i). \tag{2.3}$$

An affine function that approximates (2.3) is

$$A\hat{x} + B\hat{y} + C = y_0\hat{x} + x_0\hat{y} - x_0 y_0$$

The approximation error equals the last term in (2.3). A very pessimistic estimate is used to express the approximation error in the form of $D\varepsilon_k$,

$$(\sum_{i=1}^{N} x_i\varepsilon_i)(\sum_{i=1}^{N} y_i\varepsilon_i) \approx (\sum_{i=1}^{N} |x_i|)(\sum_{i=1}^{N} |y_i|)\varepsilon_k,$$

where ε_k is a new noise symbol. This is referred to as *trivial range estimation.* Hence the interval extension for the multiplication is

$$\hat{z} = \hat{x} \cdot \hat{y} \approx y_0\hat{x} + x_0\hat{y} - x_0y_0 + (\sum_{i=1}^{N} |x_i|)(\sum_{i=1}^{N} |y_i|)\varepsilon_k. \tag{2.4}$$

The division $z = x/y$ is computed indirectly from multiplication by

$$x/y = 1/y \cdot x$$

with similar approximation developed for the inverse $1/y$ calculation.

2.2.3 Limitations of affine arithmetic

The use of noise symbols in affine arithmetic provides a means for tracking correlations among intervals. However, it also introduces adverse effects. First, non-affine functions always introduce new noise symbols, and hence for applications that involve thousands of non-affine functions, there will be an enormous number of noise symbols being carried from one computation to another. This may significantly slow down interval computations. Second, for an affine interval with a large number of uncertainty terms, the interval's bounds determined by the conventional method (i.e., the extreme values computed by letting all noise symbols equal to ± 1) are very pessimistic, since the probability of these bounds being reached is extremely low. We will discuss this issue in more detail in Chapter 3.

Another major source of pessimism in the approximation in non-affine functions is the need to preserve the center-symmetric first-order polynomial form. As we have shown, for a non-affine unary function, not only does it rely on an approximating affine function to capture the first-order relationship between the inputs and the output, but also it conservatively assumes independence between the approximation error and the inputs. This issue is even more serious for binary non-affine functions, such as multiplication and division. As we will illustrate in Section 4.1, the conservative range estimated for multiplication could be four times the true range, and may be even worse for division.

In this dissertation, we provide improvements to affine arithmetic. There are other interval techniques, such as the centred form [77], quantile arithmetic [38] and generalized interval arith-

metic [34]. However, none of them handles correlations as well as affine arithmetic. Therefore, we do not give detailed discussion on them.

Chapter 3

Probabilistic Bounding for Affine Intervals

Although affine arithmetic is able to provide tighter bounds than interval arithmetric, it becomes very pessimistic for large scale applications. We show that abandoning conservative deterministic bounds in favor of tighter—but approximate—probabilistic bounds can help alleviate this problem. In this chapter, we first discuss the connection between an affine interval and a normal distribution. Based on this connection, we introduce a novel probabilistic bounding method that substantially reduces pessimism. Then, we apply the improved affine arithmetic to a DSP application— range and error analysis for finite-precision DSP design. By modeling variables in a DSP algorithm as affine intervals, we are able to accurately track their ranges and roundoff errors and predict the minimal bitwidth requirement for a DSP design.

3.1 Motivation

Affine arithmetic captures how the bounds of uncertainty quantities change during interval computations. Often, it is able to provide a more accurate estimate for the worst case scenario than other interval methods, since it carefully handles the correlations among uncertain quantities. However, in many practical applications, the absolute worst case is extremely rare, and hence, we are more concerned about the "probabilistic" worst case that occurs with a reasonably likely probability. For

instance, when optimizing a fixed-point FFT core, we choose a bitwidth that can represent most numbers without overflow; it would be unwise if we add more hardware resources just to accommodate the largest number that occurs only once every million computations.

The discrepancy between the absolute worst case and the probabilistic worst case can be very large. Since an affine interval models an uncertain quantity as a linear combination of many uncertain components, the absolute worst case happens only when all the uncertain components take the extreme value simultaneously, which is very unlikely to take place. We demonstrate it by the following simple example. Suppose we have an uncertain quantity x in an affine form: $\hat{x} = \sum_i^N \varepsilon_i$. Obviously the theoretical upper bound of x is N. This worst case happens when all ε_i's equal 1. However, we can conduct Monte Carlo simulation for 10^6 runs, but never encounter this case during the simulation. In fact, the maximum value that occurs in the simulation is much smaller than the upper bound. Table 3.1 compares the simulated maximum and the theoretical upper bounds, for N equal to 10, 100, 1000, and 10000. The ratios in the table suggest that the more components there are in an affine interval, the more pessimistic the upper bound becomes. When there are 10000 uncertain components, about 97% of the interval is never reached during simulation.

N	10	100	1000	10000
simulated max / theoretical bound	75%	25%	7.5%	2.9%

Table 3.1: Pessimistic bounds of the affine intervals

In real applications, there are usually a large number of uncertain components. As shown by the example, affine arithmetic is more pessimistic for large scale applications. In order to overcome this problem, we have to take probabilistic information into account while computing with intervals. The core theories of this chapter are about how we link affine intervals and probability distributions together to improve the applicability of affine arithmetic in real applications.

3.2 Core Theories

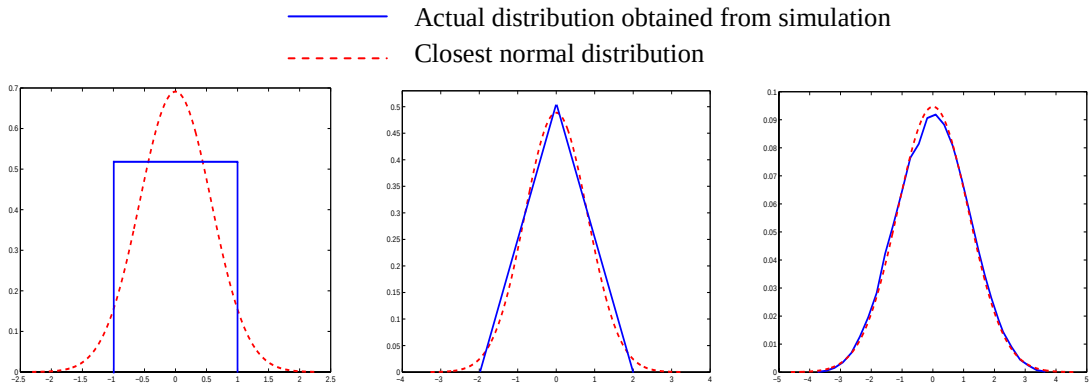
3.2.1 Affine interval and Gaussian distribution

Conventionally, intervals and explicit PDFs are two completely different representation forms for uncertain quantities. Interval techniques capture the bounds of uncertainties and propagate the bounds as interval computations are conducted. Explicit PDFs, on the other hand, emphasize on probabilistic information, i.e., how probable is the random quantity to take a certain value. Bounds are not very important to a distribution, and in fact, many well-known distributions are not even bounded, i.e., they do not have finite support.

The affine interval, however, is a special interval form that naturally implies certain probabilistic information inside the interval. Considering such information can reduce pessimism and also improve efficiency of interval computations. By definition, an affine interval describes the total uncertainty contributed by a number of independent uncertain components. Without knowing the specific distribution of each individual component, it is usually "implicitly" assumed that each component has a uniform distribution (although it could have any other distribution, for example, normal distribution). More specifically, in the affine interval definition, $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i$, each ε_i is assumed to be uniformly distributed in [-1, 1]. Interestingly, as more components are added to an affine interval, the distribution inside the affine interval rapidly converges to a normal distribution. In Figure 3.1, we show the distributions for affine intervals with an increasing number of uncertainty terms.

Such convergence in distribution is of no surprise. The theory behind it is the famous *Central Limit Theorem* [82]. It relaxes the requirement for the individual distributions to be any type, as long as they are independent and identical:

> *Let $X_1, X_2, ..., X_N$ be a set of independent identically distributed random variables with mean 0 and a finite variance σ_i^2. Let $S_N = \sum_{i=1}^{N} X_i$. We denote the standard normal distribution with $\mathcal{N}(0, 1)$. We have*
>
> $$\frac{S_N}{\sqrt{\sum_{i=1}^{N} \sigma_i^2}} \rightarrow \mathcal{N}(0, 1)$$
>
> *as N increases.*



(a) n = 1 (b) n = 2 (c) n = 4

Figure 3.1: Distribution inside an affine interval

Assume $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i$, where each ε_i is uniformly distributed in [-1, 1]. The distribution of x for $N = 1, 2, 4$ are shown by the solid lines in (a), (b), and (c), respectively. As N gets larger, the distribution converges to a normal distribution.

For the more general case, that is, when the individual components are not identically distributed, there is the *Lindeberg-Feller Central Limit Theorem* [27], which describes the convergence when the *Lindeberg condition* is satisfied:

> *Given random variables $X_1, X_2, ...$, the mean of X_i is 0 and its finite variance is σ_i. Let $S_N = \sum_{i=1}^{N} X_i$. If σ_i^2 is small compared to $\sum_{i=1}^{N} \sigma_i^2$ for all i's (referred to as the Lindeberg condition), then*
>
> $$\frac{S_N}{\sqrt{\sum_{i=1}^{N} \sigma_i^2}} \rightarrow \mathcal{N}(0,1)$$
>
> *as N increases.*

The Lindeberg condition essentially states that as long as none of the individual components dominates the sum, the convergence to a normal distribution still holds.

It is worth noting that even when the distribution inside an affine interval is sufficiently close to a normal distribution, there is still an important difference between the two: the former is bounded by finite support, while the latter has a long distribution tail, extending to infinity. Therefore when N is large, the distribution of an affine interval should be considered as a "truncated" normal distribution with finite bounds. Next, we will discuss how we use such probabilistic information to redefine the bounds of an affine interval and to reduce pessimism.

3.2.2 Probabilistic bounds for affine intervals

We are now introducing probabilistic bounds for both "1D" and "2D" affine intervals which mean a single affine interval and a pair of correlated affine intervals, respectively. As we shall see, the terminology comes from a geometric view of affine intervals, as they imply a bounded region over either the real line or the real plane.

Probabilistic bound for an 1D affine interval

An affine interval $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i$ is naturally bounded by what we call the *hard bounds*, $x_0 - \sum_{i=1}^{N} |x_i|$ and $x_0 - \sum_{i=1}^{N} |x_i|$. These bounds are reached only when all the noise symbol ε_i's simultaneously take the extreme value, 1 or -1, which is extremely rare when N is large. Indeed, as indicated by the normal distribution, most of the "mass" of the interval is close to the central value; the tails are very improbable. As a result, the probability densities at the original hard bounds are extremely low. Using these pessimistic bounds in interval computations may lead to mathematically sound, but practically overly conservative results.

Therefore we use the implication of a normal distribution and propose less pessimistic, approximate bounds for an affine interval in case of large N. It is borrowed from the concept of *confidence interval* from statistical inference [82]. A confidence interval gives the range that covers the parameter to be estimated with a specified probability, called *confidence level*, which is most often 90% or higher. A level λ confidence interval for a parameter x is given by two confidence limits $\underline{x}_\lambda$ and $\overline{x}_\lambda$ such that

$$P(\underline{x}_\lambda \leq x \leq \overline{x}_\lambda) = \lambda. \tag{3.1}$$

We use the confidence limits as the new "soft bounds" for an affine interval, and call them the *probabilistic bounds*. The original hard bounds can be viewed as confidence limits with $\lambda = 1$. Since an affine interval is centrally symmetric, the new probabilistic bounds, $\underline{x}_\lambda$ and $\overline{x}_\lambda$, should also be symmetric to the central value. Hence, we define them as a multiple of the standard deviation beyond the mean, with the standard deviation computed as $\sigma = \sqrt{\sum_{i=1}^{N} x_i^2/3}$ (suppose ε_i is uniformly

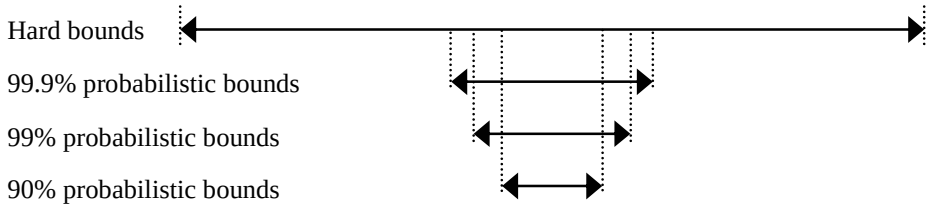


Figure 3.2: Hard bounds and probabilistic bounds

These are the hard bounds and the probabilistic bounds for $\sum_i^{50} \varepsilon_i$.

distributed in [-1, 1], and hence the variance of the term $x_i\varepsilon_i$ is $x_i^2/3$):

$$\underline{x}_\lambda = x_0 - K\sigma$$
$$\overline{x}_\lambda = x_0 + K\sigma,$$

where K is a constant that satisfies $P(x_0 - K\sigma \leq x \leq x_0 + K\sigma) = \lambda$.

The probabilistic bounding method provides more realistic and much tighter bounds than the original conservative bounding method. Here, we illustrate the difference of the two by an example $\hat{x} = \sum_{i=1}^{50} \varepsilon_i$. The hard bounds and the probabilistic bounds with different confidence levels are plotted in Figure 3.2. It is clear that slight relaxation in the confidence level can significantly reduce pessimism of the bounds.

Probabilistic bounds for a 2D affine interval

We have seen that a probabilistic interpretation for an affine interval can reduce pessimism. Often, we need to look at a pair of affine intervals jointly, e.g., when considering a binary operation on affine intervals. Next, we develop a probabilistic interpretation for a pair of affine intervals, or a 2D affine interval.

From a geometric point of view, a 2D affine interval describes a bounded region in a two dimensional space. More specifically, a 2D affine interval indicates a convex polygon, symmetric around the central point; each pair of parallel sides corresponds to a noise symbol ε_i shared by the two affine intervals. As we shall show in Section 4.2, building the corresponding polygon for a 2D affine interval is vital in our new algorithms for binary interval computations. So we first explain how to construct a polygon from a 2D affine interval, using a trivial example from from [22].

Suppose the quantities x and y are represented by

$$\hat{x} = 10 + 2\varepsilon_1 + 1\varepsilon_2 - 1\varepsilon_4$$

$$\hat{y} = 20 - 3\varepsilon_1 + 1\varepsilon_3 + 4\varepsilon_4.$$

This data tells us that x's range is [6, 14] and y's range is [12, 28]. Further, These two affine intervals are correlated through ε_1 and ε_4. The polygon is constructed based on the four bounds and the correlation between the two affine intervals. The detailed procedure includes the following four steps (depicted in Figure 3.3):

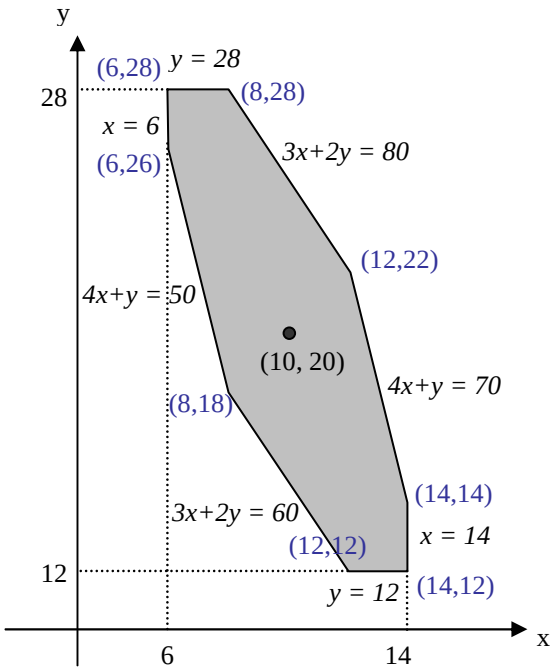


Figure 3.3: Polygon construction from a 2D affine interval

- Step 1: Find the edges implied by the upper and lower bounds of x and y. They are $x = \underline{x} = 6$, $x = \overline{x} = 14$, $y = \underline{y} = 12$, and $y = \overline{y} = 28$.

- Step 2: Find the edges implied by the shared noise symbol ε_1. The sharing of ε_1 indicates the following relationship:

$$3\hat{x} + 2\hat{y} = 70 + 3\varepsilon_2 + 2\varepsilon_3 + 5\varepsilon_4,$$

which describes a region between two lines. Fixing the noise symbols at -1 or +1 gives the boundary of this region:

$$60 \leq 3x + 2y \leq 80.$$

Therefore, the two corresponding edges of the polygon are $3x + 2y = 60$ and $3x + 2y = 80$.

- Step 3: Find the edges implied by the noise symbol ε_4. The sharing of ε_4 indicates the following relationship:

$$4\hat{x} + \hat{y} = 60 + 5\varepsilon_1 + 4\varepsilon_2 + \varepsilon_3,$$

which describes a region between two lines. Fixing the noise symbols at -1 or +1 gives the boundary of this region:

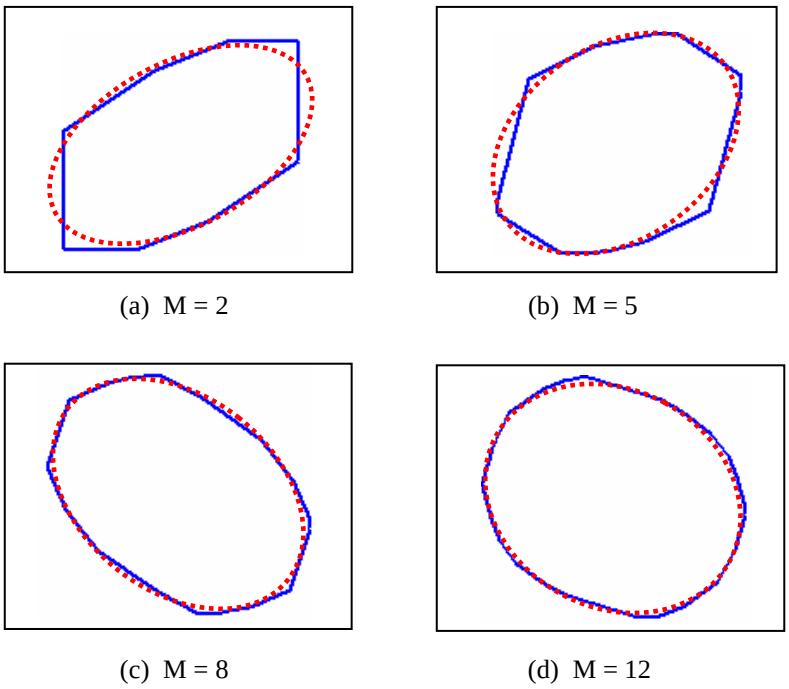
$$50 \leq 4x + y \leq 70.$$

Therefore, the two corresponding edges of the polygon are $4x + y = 50$ and $4x + y = 70$.

- Step 4: Construct the polygon from the computed edges. An important property of a convex polygon is that if we trace the perimeter of the polygon, the slopes of the edges appear in a monotone order. Therefore, once we sort the edges according to their slopes, we can easily find all the corners and construct the polygon.

From the procedure described above, it is easy to analyze the complexity of building a polygon from a 2D affine interval. Suppose there are N noise symbols, among which M symbols are shared by the two affine intervals. As we have shown in the example, the polygon has no more than $2M+4$ edges. For each edge of the polygon, the computation involves setting N noise symbols to -1 or +1, and hence takes $O(N)$ time. Therefore, finding all the edges takes $O(MN)$. Then, in order to construct the polygon from the edges, we need to sort the edges according to their slopes, which runs in $O(MlogM)$ time. So the total complexity of polygon construction is $O(MN + MlogM) = O(MN)$. In the worst case, $M = N$, and the complexity becomes $O(N^2)$. When the number of noise symbols is large, the polygon construction becomes prohibitively expensive.

Fortunately, we can approximate the polygon by an *ellipse* when a large number of noise symbols are present. A quick reasoning is that as the number of shared noise symbols increases, more edges are added to the polygon (suppose the edges from different shared noise symbols do not overlap), and consequently, the perimeter gradually approaches a smooth ellipse (see Figure 3.4).



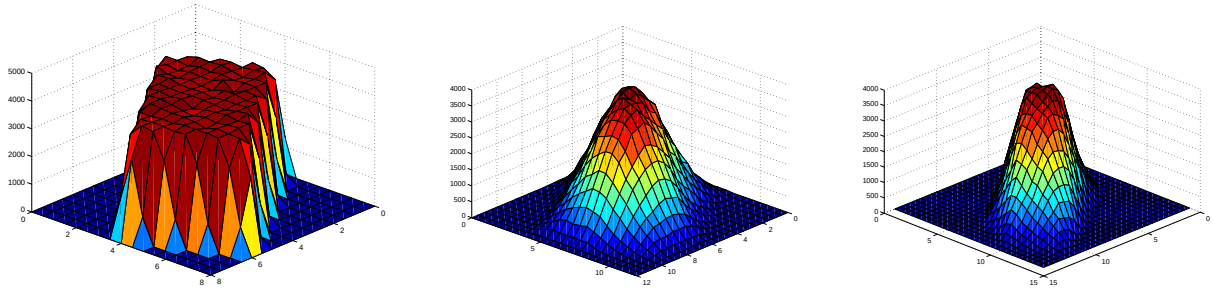
(a) M = 2 (b) M = 5

(c) M = 8 (d) M = 12

Figure 3.4: Approximating a polygon by an ellipse

As the number of shared noise symbols M increases, the corresponding polygon (solid line) becomes closer to an ellipse (dotted line).

Next, we give a more rigorous justification by looking at the probabilistic distribution of a 2D affine interval. Within the polygon, it is actually not equally probable, especially when there exists a large number of noise symbols. Figure 3.5 shows the distribution within a 2D affine interval for various N's. Analogous to the 1D case, the central limit theorem suggests that as N increases, the distribution of a 2D affine interval approaches a joint normal distribution. So distribution density is high around the center and extremely low at the perimeter of the polygon.



$N = 2$ $N = 3$ $N = 4$

Figure 3.5: Distribution inside a 2D affine interval

The distribution inside a 2D affine interval is not uniform. As the number of total noise symbols N increases, the distribution approaches a joint normal distribution.

A joint normal distribution is characterized by the following density function:

$$f_{XY}(x,y) = \frac{1}{2\pi\sigma_x\sigma_x\sqrt{1-\rho^2}} \cdot e^{\left[-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y} + \left(\frac{y-\mu_y}{\sigma_y}\right)^2\right]\right]} \tag{3.2}$$

Interestingly, a constant density curve is a rotated ellipse. If we set the density function equal to a constant and take the natural logarithm of both sides, the resulting equation describes an ellipse.

Similar to the confidence interval we present in the last section, here we propose to use *confidence ellipse* as a probabilistic description of a 2D affine interval. The confidence level, λ, is the probability with which a sample is within the ellipse. Suppose the ellipse equation is

$$\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y} + \left(\frac{y-\mu_y}{\sigma_y}\right)^2 = K(1-\rho^2), \tag{3.3}$$

where the parameter K is a function of the confidence level λ and determines the size of the ellipse. By coordinate transformation, we can compute λ as

$$\lambda = P(\text{sample is inside the ellipse}) = 1 - e^{-K/2}. \tag{3.4}$$

Detailed derivation is given in Appendix A. Therefore, for a user specified confidence level λ, K is obtained by

$$K = -2\ln(1-\lambda). \tag{3.5}$$

Constructing an ellipse from a 2D affine interval is much faster than constructing a polygon. For a 2D affine interval with N noise symbols, it takes $O(N)$ to compute the standard deviations and the correlation coefficient. It takes a constant time to compute other parameters in the ellipse equation (A.1). Therefore, the total run time complexity of building an ellipse is $O(N)$, as opposed to $O(MN)$ for building a polygon.

In addition, ellipse approximation improves efficiency of binary interval operations. Later in Section 4.2 we will show that tracing the perimeter of an ellipse is a critical step in our proposed binary interval computations. Compared to polygon tracing, which requires us to separately examine $O(M)$ line equations, ellipse tracing only need to examine one ellipse equation, leading to much more efficient interval computations.

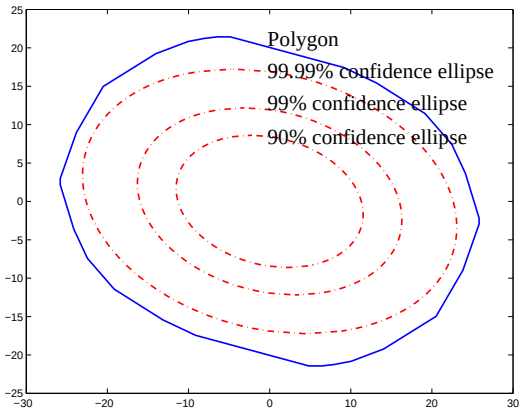


Figure 3.6: Polygon and confidence ellipses

Note: The polygon and the ellipses are constructed from a 2D affine interval with 12 noise symbols.

Another advantage of using the ellipse is that it reduces pessimism. Figure 3.6 plots a polygon and a series of ellipses with different λ's. We can see that the bounded region shrinks dramatically with slight relaxation in the confidence level. The 90% confidence ellipse is much smaller compared to the original polygon.

3.2.3 Initialization of affine intervals from given probabilistic information

In the last two sections, we have offered a probabilistic interpretation for affine intervals. In this section, we look at an opposite problem, which is, given known probabilistic information, how one can express it with affine intervals.

In interval-based analysis, we usually regard the inputs of an uncertain system as independent uncertainty sources, and model them each as an affine interval with a unique noise symbol. However, very often, these observable quantities are not independent, and are commonly modeled by a joint normal distribution. Ignoring the dependency may lead to serious inaccuracies. Hence, it is important to maintain the correlation structure, while converting them into affine intervals.

The key is to extract the independent components from the correlated quantities, which can be achieved by conducting *Principle Component Analysis*, or PCA [40]. The mathematical procedure of PCA involves performing the *singular value decomposition* to find out the eigenvalues and the eigenvectors of the covariance matrix, based on which a transformation is established between the original quantities and a set of independent components. We denote the original set of uncertain quantities by $[x_1, x_2, \ldots, x_n]'$, whose mean vector is $[\mu_1, \mu_2, \ldots, \mu_n]'$ and covariance matrix is Σ. The eigenvalues of Σ are $r_1, r_2, \ldots, r_n$, and the eigenvectors are $V_1, V_2, \ldots, V_n$. The transformation found by PCA is

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} + [V_1\ V_2\ \cdots\ V_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{3.6}$$

where y_i's are independent random variables, with zero means and standard deviation $\sigma_{y_i} = \sqrt{r_i}$.

Now, the original quantities are represented in terms of a set of independent components. To comply with the affine interval definition, we bound each y_i by $\pm 3\sigma_{y_i}$, and replace it with $3\sigma_{y_i}\varepsilon_i$, where ε_i is a normally distributed random variable with $\sigma = 1/3$ and is bounded by $[-1, 1]$. There-

fore, the original quantities are converted into affine intervals as

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} + [V_1\ V_2\ \cdots\ V_n] \begin{bmatrix} 3\sqrt{r_1} & & & \\ & 3\sqrt{r_2} & & \\ & & \ddots & \\ & & & 3\sqrt{r_n} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}, \tag{3.7}$$

and their correlation is captured by the sharing of the noise symbol ε_i's.

3.3 Application—Finite Precision Analysis for DSP Design

Affine arithmetic with probabilistic bounding finds a very attractive application in finite-precision DSP (Digital Signal Processing) design. Modern DSP applications are typically prototyped using floating-point arithmetic, which offers both large dynamic range and high precision for numerical computations. However, for hardware implementation, they are transformed into some hardware-efficient format (namely, fixed-point or custom-precision floating-point) to reduce silicon area and power consumption. The data format transformation usually distorts the natural form of the algorithm and forces awkward design tradeoffs. This time-consuming and error-prone procedure often becomes the bottleneck of the entire system design flow. Therefore, it calls for efficient techniques that can automatically analyze the algorithm distortions, or the so called *finite-precision effects*, and assist the design procedure. Affine arithmetic is an excellent candidate, as it provides a means for tracking correlated uncertainties which, in this particular case, are the adverse effects caused by the data format transformation.

In Section 3.3.1, we first offer necessary background on finite-precision effects and review the existing approaches on how to analyze them. Next, in Section 3.3.2, we outline a fast and accurate static analysis approach based on affine arithmetic with probabilistic bounding. It provides a general solution for both fixed-point and floating-point DSP design. Then, modeling details for fixed-point and floating-point arithmetic are discussed in Section 3.3.3 and Section 3.3.4, respectively. Experimental results are provided in Section 3.3.5. Finally, we demonstrate the utilization of our efficient analytic technique by two DSP applications in Section 3.3.6.

3.3.1 Background

Finite-Precision Data Formats

Two common finite-precision formats are fixed-point and floating-point formats. To better understand the design issues associated with finite-precision arithmetic, let us first briefly review these two data formats.

Fixed-point

In a fixed-point format, a real number is represented by two elements: an integer part (e.g., 3) and a fraction part (e.g., 0.14159). It is usually in sign-magnitude representation, and therefore the leading bit indicates the sign. The bitwidth of the integer part (i) determines the largest representable number (absolute value), and the bitwidth of the fraction part (f) determines the smallest non-zero number (absolute value). Precision of the representable numbers is determined also by the fraction width. Taking an example of $[i = 16, f = 16]$ fixed-point format, it can accommodate a dynamic range from 2^{-16} to 2^{16}, and the 16-bit fraction sets the resolution, or the distance between two adjacent representable numbers, to be 2^{-16}.

Since the integer part and the fraction part compete for the limited number of bits in a word, it is hard to achieve both wide dynamic range and high precision simultaneously. However, fixed-point formats are often chosen over floating-point formats by DSP designers due to their hardware simplicity.

Floating-point

Floating-point, on the other hand, is reputed for both wide dynamic range and high precision. It represents a number by three fields: sign bit (s), exponent (e) and mantissa (m) as follows:

$$x = (-1)^s \cdot 2^{e-bias} \cdot (1 + m) \tag{3.8}$$

where m is in $[0, 1)$, and the use of $bias$ is to represent both positive and negative exponent values, since e is an unsigned integer. Dynamic range is determined by the exponent bitwidth, and precision is determined by the mantissa bitwidth. The widely adopted IEEE single-precision floating-point standard uses an 8-bit exponent which can reach a dynamic range roughly from 2^{-126} to 2^{127}, and a 23-bit fraction which has a resolution of $2^{e-127} \cdot 2^{-23}$. It is worth noting that the resolution of a

floating point number is related to not only the mantissa width, but also the value of the exponent: when using the same mantissa width, smaller numbers have finer resolution.

In general, floating-point provides maximal representation capability, freeing software designers from concerning themselves with dynamic range and precision. However, hardware complexity is the bane of floating-point arithmetic. Since floating-point arithmetic operations involve separate treatments for the three fields (sign, exponent and mantissa), they are much more complicated than the fixed-point counterparts. Use the floating-point addition as an example, first, the mantissa of one operand is shifted so that the two exponents become the same, then, addition is performed on the two aligned mantissas, and finally, the sum is normalized and rounded to the required format. In each step, separate operations are performed on the three fields. It is the disadvantage in hardware that eventually hinders the utilization of floating-point unit in power-stringent mobile appliances. Lightweight floating-point [26], or custom-precision floating-point [86], has therefore been proposed to "tailor" the data format, mainly the bitwidth, to the specific application requirements and hence reduce hardware cost.

Finite-Precision Effects

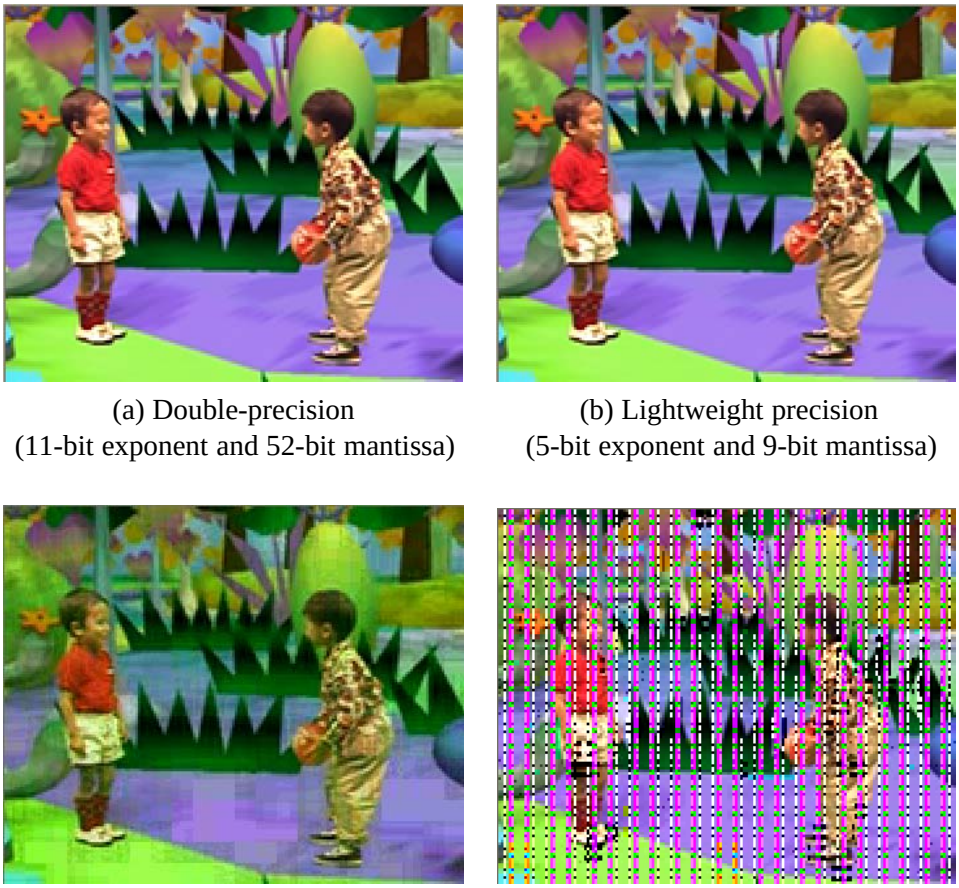
Limiting the bitwidth of data representation format causes algorithm distortions, often referred to as *finite-precision* effects. They can be divided into two distinct issues: *overflow* and *precision loss*. Overflow occurs if the integer width (for fixed-point) or the exponent width (for floating-point) is too small to cover the range that the variables may vary in a DSP algorithm. It often causes severe numerical problems and leads to system failure. Precision loss, also known as *roundoff error*, is introduced at each arithmetic operation when the result is rounded to the required number of bits. In DSP applications, precision loss are perceived as degraded system performance, such as sound quality, speech recognition rate, video quality, ..., etc. Many DSP applications have tolerance to precision loss to a certain degree. One of the reasons is that numerical precision loss under a certain threshold may not be perceptible by human eyes and ears, and hence is allowed in the final implementation.

We show in Figure 3.7 an example of finite-precision effects on a H.263 video decoder. A typical video decoder receives an encoded bit-stream, performs a series of floating-point computa-

tions, and returns, frame by frame, the decoded video. We implement the video decoding algorithm with custom floating-point and study the finite precision effects by varying the exponent and mantissa bitwidths. Figure 3.7 shows one frame of the output video sequence, decoded using different floating-point formats. Figure 3.7(a) displays a reference frame, decoded with the IEEE-standard double-precision floating-point. When we reduce the exponent bitwidth from 11 bits to 5 bits and the mantissa bitwidth from 52 bits to 9 bits, no significant video quality degradation is visible (see Figure 3.7(b)), although precision loss can be measured numerically. The bottom two figures show examples of severe precision loss and overflow. When the mantissa bitwidth is further reduced down to 4 bits and the exponent is kept at 5 bits, color distortion and blocky artifacts become perceptible (Figure 3.7 (c)). When the exponent bitwidth is further reduced to 4 bits and the mantissa is kept at 9 bits, overflow occurs, which completely destroys the decoded video (Figure 3.7 (d)).

Finite-precision effects complicate the procedure of transforming an algorithm from its initial, "infinite" precision software form into some final, finite-precision hardware form. Designers have to make careful tradeoffs between hardware cost and system performance. As shown in the case study, it is a tricky task to minimize the hardware cost of the video decoder, while preserving satisfactory video quality: if we use more than necessary bits in the data format, hardware cost may be prohibitively high; on the other hand, slight under-design (meaning reducing bitwidth too much) may yield unacceptable video quality. Moreover, as the complexity of DSP applications continues to increase, hand design that solely relies on designers' experience to choose a data format becomes a problematic approach. Therefore special techniques are needed to assist finite-precision DSP design.

Range and error analysis are commonly used to prevent overflow and control precision loss. The former aims to find the dynamic range of each variable in a program, from which the integer/exponent width can be determined, and the latter is to track the roundoff errors and predict the maximum error of the program output. Research on range and error analysis can be dated back to the 60's when the floating-point format was invented. Very recently, due to the rapidly-increasing capacity of custom hardware, custom-precision fixed-point/floating-point design draws a great deal of research attention, which proliferates new range and error analysis techniques. Next, we offer an introduction to recent work on these two topics.



(a) Double-precision
(11-bit exponent and 52-bit mantissa)

(b) Lightweight precision
(5-bit exponent and 9-bit mantissa)

(c) Severe precision loss
(5-bit exponent and 4-bit mantissa)

(d) Overflow
(4-bit exponent and 9-bit mantissa)

Figure 3.7: Finite-precision effects on decoded video

Existing work on range analysis

Understanding the dynamic range of a floating-point program is critical to finite-precision design. It helps to determine the minimum integer/exponent bitwidth that can prevent overflow. In the following, we review three major range analysis techniques: simulation-based approach, format propagation approach and range propagation approach.

Simulation-based approach

Simulation is a widely-used approach and can be found in many fixed-point and floating-point design systems [25, 26, 44, 48, 49, 52, 72]. It simulates the algorithm under question with a large number of input data sets, sampled from real application data, or randomly generated according to the input statistics. During simulation, the maximum and the minimum values of each variable are recorded, from which the required integer width can be determined. One legitimate concern of the simulated range is that it may be an overly-optimistic estimate: if we choose a sequence of input samples that are too few or incorrectly distributed, we may fail to find the extreme values that real-life use will encounter.

Format propagation approaches

Format propagation *statically* estimates the integer widths required in a program, without resorting to detailed simulation on a specific input data set. It was initially developed to determine bitwidth requirements in integer programs. In the PICO system developed by HP Lab [60], bitwidth of each integer variable is analyzed according to a set of rules, called *Opcode transfer functions*, which describe the relationship between the bitwidths of the operands (inputs) and the bitwidth of the result (output) for common arithmetic operators. For example, if the two inputs of an addition have 8 bits, then the output need no more than 9 bits. If the output of an addition is known to have 8 bits, then the two inputs need no more than 8 bits. Table 3.2 lists the Opcode transfer functions for addition and multiplication, both forward (computing the output's width based on the inputs' widths) and backward (computing the inputs' widths based on the output's width). We denote the input integer widths by s_1, s_2 and the output integer width by d. Based on these rules, the bitwidths can be propagated forward and backward through the dataflow graph. Typically, it requires the user to specify the integer bitwidth for some variables, and the integer width for the rest of the variables

can be deduced by forward and backward format propagation.

Operation	Forward function	Backward function
add	$d = max(s_1, s2) + 1$	$s_1 = d$, $s_2 = d$
multiply	$d = s_1 + s2$	$s_1 = d$, $s_2 = d$

Table 3.2: Examples of format propagation rules

s_1 and s_2 are the integer widths of the inputs, and d is the integer width of the output.

Since fixed-point behaves almost like integers, this technique is well suited for fixed-point design, with minor modifications on some propagation rules (e.g. the one for division) to model fixed-point operations. In Précis [14] and FRIDGE [43], format propagation is used to determine the integer widths needed in a fixed-point program. This technique is improved in [9] to handle parameterized loops, i.e., a loop whose number of iterations is unknown at compile time. In [19], a variation of this technique is proposed to propagate not only the integer bitwidth, but also the entire width of a fixed-point word.

Format propagation gives a quick, but conservative, estimate of the required integer bitwidths. The problem of overestimation is especially noticeable in addition. When two numbers with n bits are added, the output is always given $n + 1$ bits, while in reality, the sum of these two numbers may never need this one more bit. If there is a sequence of additions, the estimated bitwidths will grow rapidly to the maximum permissible width, giving little insight on possible bitwidth reduction. Hence, a slightly different approach—range propagation—is more commonly used.

Range propagation approach

In range propagation, variable range, expressed as a (min, max) pair, is propagated along the dataflow graph during compile time. In contrast to format propagation, range propagation requires prior knowledge on the ranges of the inputs of a program. Then, these ranges are propagated based on the rules from interval arithmetic. After a pass of forward propagation, the integer width of each variable is determined according to its range.

Interval-based range propagation and its variations are used in a few design systems. In [4], a multi-interval concept is proposed, where two intervals are used for each variable, with one for

the dynamic range in the positive region, and the other for the negative region. The purpose of using multi-interval is to capture not only the growth of the integer part (towards $\pm\infty$), but also the growth of the fraction part (towards zero). From the multi-interval deduced for each variable, we can determine the integer width that guarantees no overflow, as well as the fraction width that ensures no quantization. To alleviate the conservativeness of pure range propagation, a hybrid method that combines statistical simulation with analytical range propagation is proposed in [16] .

Compared to format propagation, range propagation takes into account more realistic information, and therefore yields more accurate estimations. Further, it is suitable for both floating-point and fixed-point design. However, it may be still too pessimistic due to an inherent problem of interval arithmetic, i.e., the incapability of handling correlations among variables. If two variables, both in the range of [-10, 10], are correlated in a way that whenever one takes a negative value, the other takes a positive value, then the sum of these two variables will never reach the two extremes, namely -20 and 20, predicted by interval arithmetic. The true resulting interval may be much narrower than [-20, 20]. But with conventional interval arithmetic, it is impossible to express such variable correlation. Attack on this issue has been made in [11] where the interval of each input of a program is divided into many subintervals. It reduces the pessimism only to a certain degree, but at the same time, the time complexity grows exponentially with the number of partitions in the intervals.

Existing work on error analysis

The second part of finite-precision design is to determine the faction/mantissa bitwidth which directly affects the achievable accuracy of an algorithm. It normally relies on an error analysis engine to find the worst-case error, or the statistics of the error, given a certain choice of bitwidth. Early work on floating-point and fixed-point error analysis is represented by Oppenheim [71, 92] and Liu's [55] study on quantization errors in digital filters. The techniques are later refined by Rao [76] and Menard [64]. Error analysis on specific DSP transforms can be found in [45, 95] (for DCT transform) and [59] (for FFT transform). A common characteristic of the above methods is that a closed-form mathematical description of the DSP system, e.g., the transfer function, is required in order to derive an analytical solution, which limits the application of these methods. The approaches we are about to discuss, however, do not have such a constraint and can be applied to any DSP algo-

rithm described using a programming language. These error analysis approaches can be classified into three categories: simulation-based approach, interval-based approach and derivative-based approach.

Simulation-based approach

A straightforward error analysis method is simulation, which has been widely adopted in commercial DSP design tools, such as SPW (Cadence) [10], CoCentric (Synopsis) [84], DSP Station (Mentor Graphics) [33], and Matlab Simulink (Mathworks) [62]. In a simulation-based design flow, two implementations of an algorithm are simulated simultaneously, one with finite-precision and the other with the IEEE double-precision. The error bound is then estimated by the maximum difference between the outputs of these two implementations, after simulating a suitably large set of inputs. Finally this error is checked against a certain application-dependent criteria, which decides whether the current format choice needs further adjustment. In our previous work [26], we have used simulation to find the minimal mantissa width for a custom floating-point video decoder. Starting from 23-bit mantissa, we gradually reduce the bitwidth, and for each bitwidth setting, we run simulation over many frames of video and check the precision loss, quantified by PSNR (Peak-Signal-to-Noise-Ratio). Larger error results in smaller PSNR measure. As shown in Figure 3.8, PSNR remains almost constant across a rather wide range of mantissa bitwidths, which means reduction in mantissa width does not noticeably degrade the decoded video quality in this range. After the cutoff point at around 9 bit, PSNR starts dropping. Thus, we reduce the mantissa bitwidth from 23 bits to 9 bits, with only 0.05dB PSNR loss, which is not perceptible to human eyes.

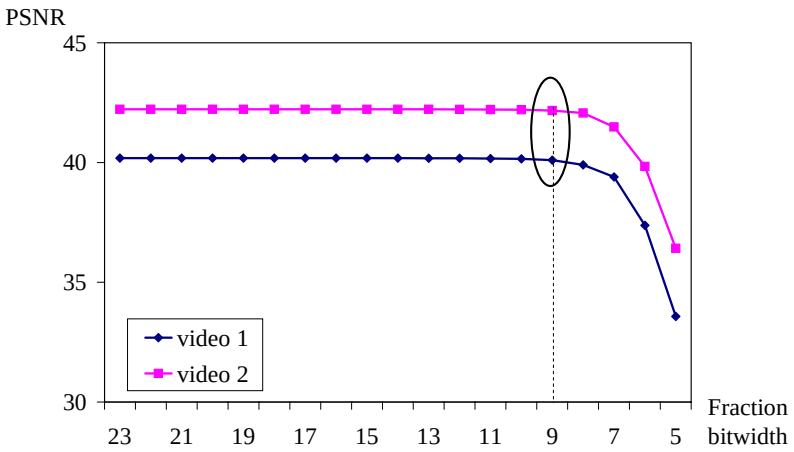


Figure 3.8: Video quality vs. fraction width

A major limitation of simulation is the high computational cost. First, since simulation is usually performed on floating-point machines, fixed-point and custom floating-point computations can only be emulated, thereby increasing the execution time by one to two orders of magnitude [64]. Second, to obtain reliable estimation, simulation has to be repeated on a large number of input samples, usually in the order of 10^5 to 10^6. Third, in design optimization, bitwidth is usually adjusted iteratively, and the time-consuming simulation is required for each bitwidth adjustment. Especially when multiple data formats are allowed in an algorithm, the search space of bitwidth settings is tremendously large, making rigorous simulation unaffordable. Therefore, static analysis techniques that can be performed at compile time are an attractive alternative for design optimization.

Interval-based approach

A classic static analysis technique is interval arithmetic. It is applied to floating-point error analysis in [47], where the error of a floating-point variable is represented by an interval. Of course, this interval gives conservative estimate to an error, meaning that the actual error that can possibly occur must lie in this interval. Based on the properties of floating-point operations, the authors developed a set of propagation functions that compute the interval of the output error for a specific arithmetic operation, given the intervals of the inputs, the operation type, and the mantissa width used during the operation. Thus, error intervals can be propagated from the inputs to the outputs through a dataflow graph. We briefly introduce the propagation rules in the following, as we will later compare the results of our proposed method against it.

Consider a binary operation $c = a \circ b$, where $a \in A$ and $b \in B$, and the errors of a and b are in the intervals $\Delta(a)$, $\Delta(b)$, respectively. The error of the output c is in the interval $\Delta(c)$. The error propagation rule is expressed as $\Delta(c) = f(A, B, \Delta(a), \Delta(b), w)$, where w is the mantissa width. For fundamental operations $\circ \in \{+, -, \cdot, /\}$, its floating-point computation $\circledcirc$ has the following property:

$$\left| \frac{a \circ b - a \circledcirc b}{a \circ b} \right| \leq \varepsilon \tag{3.9}$$

where $\varepsilon = 2^{-(w+1)}$. This states that the relative error is less than a constant that is related to the mantissa width. Replacing a and b with their interval counterparts, we can derive the error propagation rules. Table 3.3 lists the results for common arithmetic operators. We can see that the results involve interval computations, such as addition, multiplication and division. These computations

Operation	Error propagation rules
$c = a + b$	$\Delta(c) = \varepsilon(A + B) + (1 + \varepsilon)\big(\Delta(a) + \Delta(b)\big)$
$c = a - b$	$\Delta(c) = \varepsilon(A - B) + (1 + \varepsilon)\big(\Delta(a) - \Delta(b)\big)$
$c = a \cdot b$	$\Delta(c) = \varepsilon(A \cdot B) + (1 + \varepsilon)\big(A \cdot \Delta(b) + B \cdot \Delta(a) + \Delta(a) \cdot \Delta(b)\big)$
$c = a/b$	$\Delta(c) = \frac{1}{B}\Big(\Delta(a) + \big(A + \Delta(a)\big)\varepsilon\Big)$

Table 3.3: Examples of error propagation rules

Lower case stands for the variable itself, and upper case represents the corresponding interval. $\Delta(x)$ is the interval for the error of the floating-point variable x. Operation on two intervals follows interval arithmetic. $\varepsilon = 2^{-(w+1)}$, where w is the mantissa width.

follow rules in interval arithmetic.

The interval-based approach efficiently estimates an error upper bound, given the data format setting and the range information of the program inputs. However, it suffers from a common problem of interval arithmetic: it conservatively estimates the growth of error intervals, which may lead to range explosion. When considering each arithmetic operation, it is assumed that the errors of the two inputs are independent, but in fact, the errors of different variables in a DSP program are often correlated, which enables error cancellation. We give an illustrative example in Figure 3.9. The variables a and b are inversely correlated due to their dependency on x, and so are their errors. Consequently, error cancellation occurs at the operation $a + b$. By ignoring correlations, this error analysis approach may not be able to achieve a reasonably tight bound.

Derivative-based approach

Another class of error analysis technique is based on partial derivatives. As presented in [90], partial derivatives are used to relate the output error of a function to the input errors. For a binary function $f(u, v)$, assume the error in u and v are $\delta(u)$ and $\delta(v)$, respectively. The error in the result is given by Taylor series expansion as

$$\delta(f) = \frac{\partial f}{\partial u}\delta(u) + \frac{\partial f}{\partial v}\delta(v) + \frac{1}{2!}\Big[\frac{\partial^2 f}{\partial u^2}\delta(u)^2 + 2\frac{\partial^2 f}{\partial u \partial v}\delta(u)\delta(v) + \frac{\partial^2 f}{\partial v^2}\delta(v)^2\Big] + \cdots + \delta_{new} \quad (3.10)$$

where δ_{new} is an new error introduced by this operation and the rest are the errors propagated from

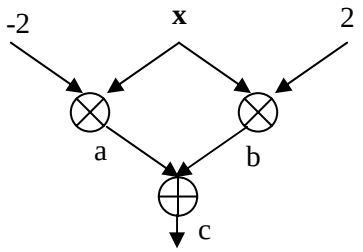


Figure 3.9: An example of error cancellation

a and b are inversely correlated through x, and so are their errors. Error cancellation happens at $a + b$.

the inputs. The partial derivative terms are evaluated at limiting values of u and v obtained from range analysis. This gives a general method for quantifying error propagation, regardless of the specific operation type. However, it is unable to take into account the correlation between u and v, and hence can be too pessimistic for some applications.

A similar technique, called *perturbation analysis*, is described in [18], targeting fixed-point nonlinear systems. It interprets the rounding error as a small perturbation on the variable, and uses Taylor series expansion to model error propagation as a linear system in response to small perturbations. In [29, 30], a technique from the optimization community, named *automatic differentiation*, is used to automatically evaluate the partial derivatives of all the functions in a dataflow graph. However, both perturbation analysis and automatic differentiation fail to consider the randomness of u and v, and only use one sample value of u and v to evaluate the partial derivatives, causing serious error underestimation.

In summary, simulation and formal methods are the two camps of range and error analysis. Simulation generally provides accurate results, if sufficient and correctly distributed input data are tested. However, it is usually a few orders of magnitude slower than formal methods, making finite-precision design a time-consuming task. Formal methods, on the other hand, are fast and do not rely on specific input data set. In addition, the results usually indicate the contributions of various parameters, giving more instructive information to design optimization. However, it often gives too pessimistic results, mainly due to the incapability of handling correlations.

3.3.2 Solution Outline

Next, we introduce a novel, fast and accurate formal method based on affine arithmetic with probabilistic bounding. It is distinguished from the prior approaches by the following key features:

- It is a *static analysis* approach. Range and error estimation is completed within a *single* pass of "simulating" the computational network. Unlike a single pass of Monte Carlo simulation in which the input values are known during run time, static analysis has only input range information available and hence performs a single pass of *interval simulation*.
- Thanks to affine arithmetic, it efficiently handles correlations among intervals, and thus significantly improves the quality of the estimated bounds.
- It offers a uniform solution for range and error analysis, applicable to both fixed-point and custom floating-point design.

Before we dive into detailed discussion, we first explain the two types of uncertainties in finite-precision DSP design. The inputs of a DSP algorithm, or the external signals, usually take values in a bounded range. Each algorithm execution may sample at a random point in this range, giving rising to the first type of uncertainty—*input value uncertainty*. The second type of uncertainty, which is specific to finite-precision programs, is *roundoff error uncertainty*. It is introduced by rounding of both external signals and internal arithmetic operations. To account for all possible algorithm executions, static analysis seeks to push the uncertain inputs and their associated roundoff errors, modeled as intervals, through the computational network, and evaluate the range of the output and its error. Very often, these uncertain operands and their errors have complicated dependencies among each other, making them difficult to track precisely.

Affine arithmetic (AA) is an efficient interval technique that preserves rich information about uncertainties, including sources of uncertainties and their correlations (strictly speaking, their linear correlations). It nicely meets our needs in finite-precision range and error analysis. We model a finite-precision operand x as an affine interval. More specifically, we divide its affine interval representation into two parts to separately capture value and error uncertainties: one that indicates the range of its value ($\hat{x}^r$) and the other representing the range of its roundoff error ($\hat{x}^e$). Since a

roundoff error can be seen as an additive noise on a signal, we have

$$\hat{x} = \hat{x}^r + \hat{x}^e$$

where both $\hat{x}^r$ and $\hat{x}^e$ are affine intervals, detailed as $\hat{x}^r = x_0^r + \sum_{i=1}^{n} x_i^r \varepsilon_i$ and $\hat{x}^e = x_0^e + \sum_{i=1}^{m} x_i^e \varepsilon_i$, respectively. Through the sharing of noise symbols, two types of correlation relationships are captured in this formulation. One is the dependency of an operand's roundoff error on its value, indicated by the symbol sharing between $\hat{x}^e$ and $\hat{x}^r$, and the other is the correlation between different operands, characterized by the symbol sharing between $\hat{x}_i$ and $\hat{x}_j$. AA-based static analysis computes an affine interval pair $(\hat{x}^r, \hat{x}^e)$ for every variable in a finite-precision DSP algorithm, while preserving the sharing of noise symbols.

There are three essential steps in analyzing a DSP algorithm with affine intervals:

- **Input modeling**: DSP algorithm's inputs are usually distributed in bounded ranges. For example, the DCT (Discrete Cosine Transform) in image processing takes image pixels as inputs which are known to be in [0, 255]. A trivial task is to express this range information as affine intervals. However, to accurately model real-life data, correlation among inputs has to be carefully taken into account. We perform PCA (Principle Component Analysis) on the inputs and convert the correlation into the sharing of noise symbols in affine intervals.[1] Further, this step models the roundoff error of an input as an affine interval $\hat{x}^e$, and precisely captures its dependency on $\hat{x}^r$, if there is any.

- **Interval propagation**: When affine intervals are pushed through arithmetic operations, their value and error uncertainties interact with each other, resulting in accumulation and canceling effects. We have established computation models for common finite-precision arithmetic operators, such as $\pm$, $\times$, $\div$, ..., etc, to capture uncertainty propagation and compute a new affine interval pair for the operator's output (shown in Figure 3.10). Thus, by a single traversing of the computational network, we obtain an affine interval pair that captures the value and error uncertainties of the system output.

[1] Strictly speaking, static analysis should be independent of inputs. In our case, correlation is studied on a representative set of input data, and therefore the approach is "semi-static".

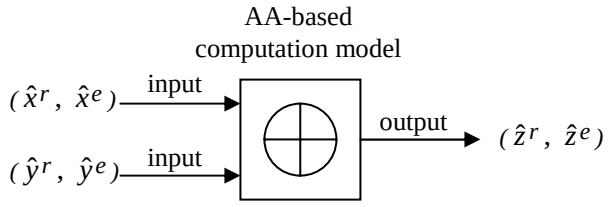


Figure 3.10: AA-based computation model

- **Bound estimation**: Finally, we use the probabilistic bounding method introduced in the previous section to estimate the bounds for the output's value and roundoff error.

A major advantage of AA-based error analysis is that it efficiently handles correlations among roundoff errors which have thwarted all earlier attempts at accurate interval-based error analysis. Figure 3.11 is a simple example of interval-based error analysis. It compares the result by AA with that by ordinary interval arithmetic (IA). At each arithmetic operator, input errors are propagated to the output, and at the same time, a new roundoff error is introduced. The AA-based intervals carry information about the uncertainty sources, depicted by various shading patterns. It enables error cancellation (happens at the last addition) and hence yields tighter interval. On the other hand, the IA-based intervals always accumulate, inevitably leading to overestimation.

3.3.3 Fixed-Point Range and Error Analysis via Affine Arithmetic

We first consider range and error analysis for fixed-point, as it is the simpler case among the two finite-precision data formats. In this section, we first provide affine form models for fixed-point operands, and then develop computation models for common fixed-point arithmetic operators.

Affine forms for fixed-point operands

Let us first isolate error uncertainty from other uncertainties. Consider a real number whose value is known to be x. In other words, x does not have any uncertainty in its value. Assume x is represented by the (i, f) fixed-point format, where i is the bitwidth for the integer part, and f is the bitwidth for the fraction part. The roundoff error is usually considered as a uniformly distributed random variable. The bound of the roundoff error in x is related to the rounding mode. If real rounding,

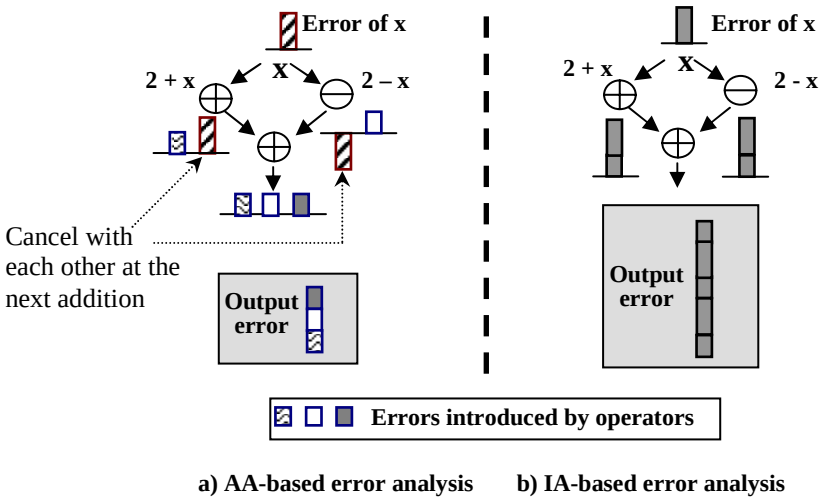


Figure 3.11: Example: comparison of AA- and IA-based error analysis

(a) shows error propagation modeled by AA. Round-off error uncertainties for each operand are represented by vertical bars. The shading pattern of an error bar indicates the source of its uncertainty. Sharing of the same error source enables error cancellation. (b) shows error propagation modeled by IA. No information on error sources is available and hence error cancellation is impossible.

or round-to-nearest, is used, then the error is bounded by $2^{-(f+1)}$. If truncation is used, then the error bound is 2^{-f}. The error models we present throughout this chapter assume real rounding. So the fixed-point representation of this number has a central value at x, and an uncertain error term, bounded by $2^{-(f+1)}$. This essentially gives the following affine interval:

$$x_f = x + 2^{-(f_x+1)}\varepsilon, \quad \text{with } \varepsilon \in [-1, 1]. \tag{3.11}$$

Note that the roundoff error is bounded by a constant that depends on its fraction-width, and is not related to its value x. Although this modeling step is incredibly simple, it reveals a fundamental difference with an ordinary interval, i.e., the use of the noise symbol ε which will be kept throughout static analysis.

Now let us incorporate value uncertainty into the equation. In the case where x's value is unknown and lies in a certain range, $[x_0 - x_1, x_0 + x_1]$, we replace x by an interval $x_0 + x_1\varepsilon_1$. Fortunately, for fixed-point, this added uncertainty does not affect the bound of the roundoff error. Therefore, an affine interval representation for a fixed-point operand is

$$\hat{x} = x_0 + x_1\varepsilon_1 + 2^{-(f_x+1)}\varepsilon_2 \tag{3.12}$$

where the value uncertainty $x_1\varepsilon_1$ and the error uncertainty $2^{-(f_x+1)}\varepsilon_2$ are two independent random terms. For illustration purposes, we separate these two types of uncertainties into two affine intervals, and use an affine interval pair ($\hat{x}^r$, $\hat{x}^e$) to represent a fixed-point operand:

$$(\hat{x}^r\ ,\ \hat{x}^e) = (x_0 + x_1\varepsilon_1\ ,\ 2^{-(f_x+1)}\varepsilon_2). \tag{3.13}$$

where both $\hat{x}^r$ and $\hat{x}^e$ have one uncertainty term, and $\hat{x}^e$'s central value is at zero. This is the affine model for the fixed-point inputs of a DSP algorithm.

However, the above model in (3.13) is not general enough for the internal fixed-point operands in a computational network. For internal operands, there is usually more than one source of value uncertainty in $\hat{x}^r$, caused by uncertainties from multiple inputs. Similarly, there usually exist multiple error uncertainty terms, because roundoff errors produced by prior arithmetic operations are accumulated into its total error. Further, the central value of $\hat{x}^e$ may not be zero. This can be easily explained by an example. Suppose a fixed-point operator is a constant 1.256. When it is represented with 2-bit fraction, the closest representable number is 1.25, and therefore the roundoff error is 0.006, which is not centered around zero. For a DSP algorithm that involves constants as operands, roundoff errors of intermediate variables tend to shift away from zero. Therefore, a general affine form representation for a fixed-point operand, is

$$(\hat{x}^r\ ,\ \hat{x}^e) = (x_0^r + \sum_{i=1}^{n} x_i^r\varepsilon_i\ ,\ x_0^e + \sum_{i=1}^{m} x_i^e\varepsilon_i). \tag{3.14}$$

In this general form, $\hat{x}^e$ may share noise symbols with $\hat{x}^r$, although in the special case in (3.13), they are independent. We will soon see how the dependency is introduced by arithmetic operators. A noise symbol may also be shared by multiple fixed-point affine intervals, which captures the common variation sources among different fixed-point operands and keeps track of their correlations.

It is worth mentioning another special case of (3.14). If a fixed-point operand is a constant c, the fixed-point affine interval in (3.14) will be reduced to

$$(\hat{c}^r\ ,\ \hat{c}^e) = (c_0^r\ ,\ c_0^e). \tag{3.15}$$

Note that there is no uncertainty terms in a constant operand.

Fixed-point computation models using AA

In order to symbolically simulate a DSP algorithm with fixed-point affine intervals as inputs, we must have computation models that replace each elementary operation on real numbers with the corresponding operation on fixed-point affine intervals, returning a fixed-point affine interval.

Let us consider specifically a binary operation, $z \leftarrow f(x, y)$ or $z \leftarrow f(x, c)$, where x and y are variables, and c is a constant. The corresponding finite-precision affine interval operation $\hat{z} \leftarrow \hat{f}(\hat{x}, \hat{y})$ or $\hat{z} \leftarrow \hat{f}(\hat{x}, \hat{c})$ computes an affine form for z which is a function of the operands' affine forms and a newly introduced roundoff error. The input interval operands, $\hat{x}$, $\hat{y}$, and $\hat{c}$, are in the form of affine interval pairs:

$$\hat{x} = \hat{x}^r + \hat{x}^e \; ; \quad (\hat{x}^r \; , \; \hat{x}^e) = (x_0^r + \sum_{i=1}^{n} x_i^r \varepsilon_i \; , \; x_0^e + \sum_{i=1}^{m} x_i^e \varepsilon_i)$$

$$\hat{y} = \hat{x}^r + \hat{x}^e \; ; \quad (\hat{y}^r \; , \; \hat{y}^e) = (y_0^r + \sum_{i=1}^{n} y_i^r \varepsilon_i \; , \; y_0^e + \sum_{i=1}^{m} y_i^e \varepsilon_i)$$

$$\hat{c} = \hat{x}^r + \hat{x}^e \; ; \quad (\hat{c}^r \; , \; \hat{c}^e) = (c_0^r \; , \; c_0^e).$$

The output $\hat{z}$ is

$$\hat{z} = f(\hat{x}, \hat{y}) + z_k^e \varepsilon_k \quad \text{or} \quad f(\hat{x}, \hat{c}) + z_k^e \varepsilon_k \tag{3.16}$$

where $z_k^e \varepsilon_k$ is a new error uncertainty term, independent of other uncertainty terms involved in the computation. The bound of this new error is related to the fraction-widths f_x, f_y, f_c and f_z. The main job is to replace the righthand side of (3.16) with an affine interval that not only captures the range and the roundoff error of z, but also preserves as much information as possible about the relationship between the output and the inputs. In other words, the goal is to find the following fixed-point affine interval for the output $\hat{z}$:

$$(\hat{z}^r \; , \; \hat{z}^e) = (z_0^r + \sum_{i=1}^{n} z_i^r \varepsilon_i \; , \; z_0^e + \sum_{i=1}^{m} z_i^e \varepsilon_i + z_k^e \varepsilon_k),$$

that relates the coefficients, z_i^r's and z_i^e's, to those of the input affine intervals.

Affine operations

For affine functions, namely $x \pm y$, cx, and $x \pm c$, developing AA-based computation models is relatively easy, because exact combination of the input affine forms $f(\hat{x}, \hat{y})$ gives an affine form.

The existence of the new roundoff error depends on the fraction-width of z. For operation $x \pm y$, a roundoff error is introduced only when z has a smaller fraction-width than either of the operands' fraction-widths. For multiplication cx, suppose the two inputs have fraction-widths f_x and f_c, an ideal multiplication yields at most ($f_x + f_c$)-bit fraction. When the fraction-width of z is smaller than $f_x + f_c$, this operation produces a non-zero roundoff error. The newly introduced roundoff error is bounded by $2^{-(f_z+1)}$.

Here we give details on addition, addition with a constant, and multiplication with a constant:

- Addition $z = x \pm y$:

$$\begin{aligned} \hat{z}^r &= \hat{x}^r \pm \hat{y}^r \\ \hat{z}^e &= \hat{x}^e \pm \hat{y}^e + z_k^e \varepsilon_k \end{aligned} \tag{3.17}$$

  where

$$z_k^e = \begin{cases} 2^{-(f_z+1)}, & f_z < max(f_x, f_y) \\ 0, & \text{otherwise} \end{cases}$$

- Addition with a constant $z = x \pm c$:

$$\begin{aligned} \hat{z}^r &= \hat{x}^r \pm c_0^r \\ \hat{z}^e &= \hat{x}^e \pm c_0^e + z_k^e \varepsilon_k \end{aligned} \tag{3.18}$$

  where

$$z_k^e = \begin{cases} 2^{-(f_z+1)}, & f_z < max(f_x, f_c) \\ 0, & \text{otherwise} \end{cases}$$

- Multiplication with a constant $z = cx$:

$$\begin{aligned} \hat{z}^r &= c_0^r \hat{x}^r \\ \hat{z}^e &= (c_0^r + c_0^e)\hat{x}^e + c_0^e \hat{x}^r + z_k^e \varepsilon_k \end{aligned} \tag{3.19}$$

  where

$$z_k^e = \begin{cases} 2^{-(f_z+1)}, & f_z < f_x + f_c \\ 0, & \text{otherwise} \end{cases}.$$

Interestingly, in (3.19), the intervals $\hat{z}^r$ and $\hat{z}^e$ are correlated through dependency on $\hat{x}^r$, while in (3.17) and (3.39), no correlation is introduced.

We can see from (3.17)–(3.19) that the exact results for these affine functions are already in affine forms, and therefore no further approximation is necessary. For non-affine functions, this nice property no longer exists, and hence, a more sophisticated mechanism is needed to approximate the results into affine forms.

Multiplication

A common non-affine function is multiplication xy. Similar to affine operations, a new roundoff error may be introduced, depending on the fraction-width. Exact finite-precision multiplication of two fixed-point affine intervals yields the following:

$$\begin{aligned}\hat{z} &= \hat{x}\hat{y} + z_k^e\varepsilon_k \\ &= (\hat{x}^r + \hat{x}^e)(\hat{y}^r + \hat{y}^e) + z_k^e\varepsilon_k \\ &= \hat{x}^r\hat{y}^r + \hat{x}^r\hat{y}^e + \hat{y}^r\hat{x}^e + \hat{x}^e\hat{y}^e + z_k^e\varepsilon_k\end{aligned} \tag{3.20}$$

where

$$z_k^e = \begin{cases} 2^{-(f_z+1)}, & f_z < f_x + f_y \\ 0, & \text{otherwise} \end{cases}.$$

It is clear that (3.20) is not in an affine form, since the product of two affine forms is a quadratic polynomial.

The challenge is to reduce the four quadratic components in (3.20), $\hat{x}^r\hat{y}^r$, $\hat{x}^r\hat{y}^e$, $\hat{y}^r\hat{x}^e$, and $\hat{x}^e\hat{x}^e$, to affine forms, while capturing the worst case variation and preserving as much correlation structure as possible. Among these four components, the first one contributes to the range of z, or $\hat{z}^r$, and the remaining three contribute to the error of z, or $\hat{z}^e$. Based on their distinctive roles in the output's uncertainty, we design different heuristics to reduce them to affine forms.

The first component in (3.20), $\hat{x}^r\hat{y}^r$, can be expanded to

$$\begin{aligned}\hat{x}^r\hat{y}^r &= (x_0^r + \sum_{i=1}^{n} x_i^r\varepsilon_i)(y_0^r + \sum_{i=1}^{n} y_i^r\varepsilon_i) \\ &= x_0^r y_0^r + \sum_{i=1}^{n}(y_0^r x_i^r + x_0^r y_i^r)\varepsilon_i + \sum_{i=1}^{n} x_i^r\varepsilon_i \sum_{j=1}^{n} y_j^r\varepsilon_j\end{aligned} \tag{3.21}$$

whose last term is a quadratic polynomial of the noise symbols. Following the same approximation scheme as in affine arithmetic, we reduce the quadratic term to a linear term

$$\sum_{i=1}^{n} x_i^r\varepsilon_i \sum_{j=1}^{n} y_j^r\varepsilon_j \approx B(\sum_{i=1}^{n} x_i^r\varepsilon_i)B(\sum_{j=1}^{n} y_j^r\varepsilon_j)\varepsilon_t,$$

where the bounding operator $B(\cdot)$ returns the probabilistic upper bound. Now, $\hat{x}^r\hat{y}^r$ is turned into an affine interval.

$$\hat{x}^r\hat{y}^r \approx x_0^r y_0^r + \sum_{i=1}^{n}(y_0^r x_i^r + x_0^r y_i^r)\varepsilon_i + B(\sum_{i=1}^{n} x_i^r\varepsilon_i)B(\sum_{j=1}^{n} y_j^r\varepsilon_j)\varepsilon_t \tag{3.22}$$

This approximation sacrifices the correlation between $\hat{x}^r\hat{y}^r$ and the quadratic terms, and "pretends" the variation from the quadratic terms is caused by a single new uncertainty source ε_t.

The second and the third components in (3.20) add to the error of z. In order to keep the possibility for error cancellation in subsequent computations, we try to preserve the original error noise symbols in the approximated affine form. Hence,

$$\hat{x}^r\hat{y}^e + \hat{y}^r\hat{x}^e \approx B(\hat{x}^r)\hat{y}^e + B(\hat{y}^r)\hat{x}^e. \tag{3.23}$$

This is a linear combination of two affine intervals, and hence is an affine interval.

Finally, the last quadratic component in (3.20), $\hat{x}^e\hat{y}^e$, is the product of two roundoff errors. It is relatively small in magnitude, compared to other terms, and thus is neglected.

Combining (3.20), (3.22) and (3.23), we obtain the following AA-based fixed-point multiplication model:

$$\begin{aligned}
\hat{z}^r &= x_0^r y_0^r + \sum_{i=1}^{n}(y_0^r x_i^r + x_0^r y_i^r)\varepsilon_i + B(\sum_{i=1}^{n} x_i^r\varepsilon_i)B(\sum_{j=1}^{n} y_j^r\varepsilon_j)\varepsilon_t \\
\hat{z}^e &= B(\hat{x}^r)\hat{y}^e + B(\hat{y}^r)\hat{x}^e + z_k^e\varepsilon_k \\
&= \sum_{i=1}^{m}(B(\hat{x}^r)y_i^e + B(\hat{y}^r)x_i^e)\varepsilon_i + z_k^e\varepsilon_k
\end{aligned} \tag{3.24}$$

where

$$z_k^e = \begin{cases} 2^{-(f_z+1)}, & f_z < f_x + f_y \\ 0, & \text{otherwise} \end{cases}.$$

By approximation, the final error interval is a linear combination of the two input error intervals and a new roundoff error. Note that the correlation between the error $\hat{z}^e$ and the range $\hat{z}^r$ is captured by the sharing of ε_i's in (3.24).

Division

To derive the model for division $z \leftarrow \frac{x}{y}$, we assume the range of y does not include zero. Generally speaking, division always introduces a roundoff error, regardless of the fraction-width.

Therefore,

$$\begin{aligned}\hat{z} &= \frac{\hat{x}}{\hat{y}} + 2^{-(f_z+1)}\varepsilon_k \\ &= \frac{\hat{x}^r + \hat{x}^e}{\hat{y}^r + \hat{y}^e} + 2^{-(f_z+1)}\varepsilon_k \\ &= (\frac{\hat{x}^r + \hat{x}^e}{\hat{y}^r})(1 + \frac{\hat{y}^e}{\hat{y}^r})^{-1} + 2^{-(f_z+1)}\varepsilon_k.\end{aligned} \tag{3.25}$$

Since values in the error interval $\hat{y}^e$ are generally much smaller than values in $\hat{y}^r$, we can approximate (3.25) to

$$\begin{aligned}\hat{z} &\approx (\frac{\hat{x}^r + \hat{x}^e}{\hat{y}^r})(1 - \frac{\hat{y}^e}{\hat{y}^r}) + 2^{-(f_z+1)}\varepsilon_k \\ &= \frac{\hat{x}^r}{\hat{y}^r} - \frac{\hat{x}^r}{(\hat{y}^r)^2}\hat{y}^e + \frac{1}{\hat{y}^r}\hat{x}^e + 2^{-(f_z+1)}\varepsilon_k.\end{aligned} \tag{3.26}$$

By applying the same approximation as in [22], we can reduce $\frac{\hat{x}^r}{\hat{y}^r}$, $\frac{\hat{x}^r}{(\hat{y}^r)^2}$, $\frac{1}{\hat{y}^r}$ to affine forms [2]. We skip the detailed derivation and denote the approximated affine form for these three terms by $\hat{f}(\frac{\hat{x}^r}{\hat{y}^r})$, $\hat{f}(\frac{\hat{x}^r}{(\hat{y}^r)^2})$, and $\hat{f}(\frac{1}{\hat{y}^r})$. Now, (3.26) becomes

$$\hat{z} \approx \hat{f}(\frac{\hat{x}^r}{\hat{y}^r}) - \hat{f}(\frac{\hat{x}^r}{(\hat{y}^r)^2})\hat{y}^e + \hat{f}(\frac{1}{\hat{y}^r})\hat{x}^e + 2^{-(f_z+1)}\varepsilon_k. \tag{3.27}$$

(3.27) can be further approximated to an affine interval using the same probabilistic bounding operator as in the multiplication model:

$$\hat{z} = \hat{f}(\frac{\hat{x}^r}{\hat{y}^r}) - B(\hat{f}(\frac{\hat{x}^r}{(\hat{y}^r)^2}))\hat{y}^e + B(\hat{f}(\frac{1}{\hat{y}^r}))\hat{x}^e + 2^{-(f_z+1)}\varepsilon_k. \tag{3.28}$$

The range and the error can be separated as the following:

$$\begin{aligned}\hat{z}^r &= \hat{f}(\frac{\hat{x}^r}{\hat{y}^r}) \\ \hat{z}^e &= B(\hat{f}(\frac{\hat{x}^r}{(\hat{y}^r)^2}))\hat{y}^e + B(\hat{f}(\frac{1}{\hat{y}^r}))\hat{x}^e + 2^{-(f_z+1)}\varepsilon_k.\end{aligned} \tag{3.29}$$

Similar to the multiplication model, we approximate the error interval as a linear combination of the input error intervals and a new roundoff error interval.

[2] According to [22], the reciprocal of an affine interval and the product of two affine intervals can be approximated to another affine interval using their *min-range approximation* and *minimax approximation*, respectively. Therefore, the quotient $\frac{\hat{x}^r}{\hat{y}^r}$ can be turned into an affine interval by a reciprocal function $\frac{1}{\hat{y}^r}$ and then a multiplication $\hat{x}^r \cdot \frac{1}{\hat{y}^r}$, and $\frac{\hat{x}^r}{(\hat{y}^r)^2}$ can be turned into an affine interval in three steps: a multiplication $(\hat{y}^r)^2$, a reciprocal function $\frac{1}{(\hat{y}^r)^2}$, and then another multiplication $\hat{x}^r \cdot \frac{1}{(\hat{y}^r)^2}$.

Summary of computation models

We have presented AA-based computation models for common fixed-point arithmetic operators. Modeling affine functions ($x \pm y$, cx, and $x \pm c$) is effortless. The resulting affine interval is a simple combination of the input intervals, in conjunction with a new uncertainty terms representing the roundoff error. The development of computation models for non-affine functions (xy and x/y) employs conservative approximations, in order to preserve a consistent first order affine form. More specifically, it repeatedly uses the probabilistic bounding operator $B(\cdot)$ to reduce quadratic terms to a linear term, thereby introducing overestimation. Therefore our AA-based static analysis is better suited for applications dominant by affine operations. Fortunately, this is the case with most DSP applications.

3.3.4 Floating-Point Range and Error Analysis via Affine Arithmetic

Floating-point error analysis, in essence, is similar to fixed-point error analysis. They both model operands as correlated affine intervals, and rely on AA-based computation models to propagate intervals from the inputs to the outputs of a computational network. However, floating-point's distinctive data representation complicates roundoff error modeling. In this section, we first introduce affine forms for floating-point operands, and then present the results for AA-based computation models, emphasizing the differences with fixed-point static analysis.

Affine forms for floating-point operands

One important property of floating-point arithmetic is that the rounding error depends not only on the mantissa bitwidth, but also on the magnitude of the operand. In floating-point representation, if a real number, whose value is known to be x, has f-bit mantissa, then its roundoff error is a random variable, bounded by $x \cdot 2^{-(f+1)}$ [93], compared to $2^{-(f+1)}$ in fixed-point representation. So the floating-point representation of this number has a central value at x and an uncertain error term that is proportional to x:

$$x_f = x + x \cdot 2^{-(f_x+1)}\varepsilon, \quad \text{with } \varepsilon \in [-1, 1]. \tag{3.30}$$

Note that the bound of the roundoff error is related not only to the mantissa-width, but also to its value x.

Next, let us consider an input operand whose value is distributed in $[x_0 - x_1, x_0 + x_1]$. First, we replace x in (3.30) with $x_0 + x_1\varepsilon_1$, and accordingly, we get

$$\hat{x} = x_0 + x_1\varepsilon_1 + (x_0 + x_1\varepsilon_1) \cdot 2^{-(f_x+1)}\varepsilon_2. \tag{3.31}$$

Unlike fixed-point, the added value uncertainty is multiplicative on the error uncertainty, which poses a challenge on our AA-based analysis, since now we have a quadratic term $(x_0 + x_1\varepsilon_1) \cdot 2^{-(f_x+1)}\varepsilon_2$. To reduce (3.31) to an affine form, we apply a probabilistic approximation, that is, to bound the roundoff error by $B(x_0 + x_1\varepsilon_1) \cdot 2^{-(f_x+1)}$. However, by applying the probabilistic bounding operator, we remove the noise symbol ε_1 from the roundoff error, and hence distort the dependency between the error uncertainty and ε_1. It is a tradeoff that we make between convenient modeling and estimation accuracy. Now (3.31) is turned to an affine interval:

$$\hat{x} = x_0 + x_1\varepsilon_1 + B(x_0 + x_1\varepsilon_1) \cdot 2^{-(f_x+1)}\varepsilon_2. \tag{3.32}$$

We further separate the range and the error as the following:

$$\begin{aligned}(\hat{x}^r\ ,\ \hat{x}^e) &= (x_0 + x_1\varepsilon_1\ ,\ B(x_0 + x_1\varepsilon_1) \cdot 2^{-(f_x+1)}\varepsilon_2) \\ &= (x_0 + x_1\varepsilon_1\ ,\ B(\hat{x}^r) \cdot 2^{-(f_x+1)}\varepsilon_2).\end{aligned} \tag{3.33}$$

Similar to the fixed-point models, (3.33) is the form only for the inputs of a computational network. A more general form for any floating-point operand is

$$(\hat{x}^r\ ,\ \hat{x}^e) = (x_0^r + \sum_{i=1}^{n} x_i^r\varepsilon_i\ ,\ x_0^e + \sum_{i=1}^{m} x_i^e\varepsilon_i). \tag{3.34}$$

Floating-point computation models using AA

Simulating a floating-point program with intervals as inputs requires new arithmetic operators that capture both interval growth and floating-point error propagation. Similar to the fixed-point error modeling, the inputs of an floating-point arithmetic operator are in the form of affine interval pairs, and the goal is to develop an output interval in the same form for common arithmetic computations.

Consider a binary operation, $z \leftarrow f(x, y)$ or $z \leftarrow f(x, c)$, where x and y are variables, and c is a constant. In our error analysis, the corresponding interval operands $\hat{x}$, $\hat{y}$, and $\hat{c}$ are in the following form:

$$\begin{aligned}
\hat{x} &= \hat{x}^r + \hat{x}^e\ ; \quad (\hat{x}^r\ ,\ \hat{x}^e) = (x_0^r + \sum_{i=1}^{n} x_i^r \varepsilon_i\ ,\ x_0^e + \sum_{i=1}^{m} x_i^e \varepsilon_i) \\
\hat{y} &= \hat{x}^r + \hat{x}^e\ ; \quad (\hat{y}^r\ ,\ \hat{y}^e) = (y_0^r + \sum_{i=1}^{n} y_i^r \varepsilon_i\ ,\ y_0^e + \sum_{i=1}^{m} y_i^e \varepsilon_i) \\
\hat{c} &= \hat{x}^r + \hat{x}^e\ ; \quad (\hat{c}^r\ ,\ \hat{c}^e) = (c_0^r\ ,\ c_0^e).
\end{aligned}$$

A floating-point arithmetic operation on intervals returns $\hat{z}$, which is a function of the input intervals plus a new roundoff error:

$$\hat{z} = f(\hat{x}, \hat{y}) + z_k^e \varepsilon_k. \tag{3.35}$$

In (3.35), the new floating-point roundoff error, $z_k^e \varepsilon_k$, is a random variable independent of other uncertainties in $\hat{z}$. Unlike fixed-point roundoff error, floating-point roundoff error always exists, regardless of the mantissa-widths of the inputs and the output. Moreover, the error bound $\frac{e}{k}$ is dependent on the result of this computation, $f(\hat{x}, \hat{y})$.

Same as in fixed-point error analysis, we strive to turn (3.35) into the following general form:

$$\hat{z} = \hat{z}^r + \hat{z}^e\ ;\ (\hat{z}^r\ ,\ \hat{z}^e) = (z_0^r + \sum_{i=1}^{n} z_i^r \varepsilon_i\ ,\ z_0^e + \sum_{i=1}^{m} z_i^e \varepsilon_i + z_k^e \varepsilon_k),$$

where the only difference with the corresponding fixed-point computation model is in the newly introduced roundoff error term $z_k^e \varepsilon_k$. For any floating-point computation, the bound of the roundoff error introduced by this operator is proportional to the computation result. Therefore, the roundoff error can be written as

$$\begin{aligned}
z_k^e \varepsilon_k &= \hat{z}^r \cdot 2^{-(f_z+1)} \varepsilon_k \\
&= (z_0^r + \sum_{i=1}^{n} z_i^r \varepsilon_i) \cdot 2^{-(f_z+1)} \varepsilon_k
\end{aligned} \tag{3.36}$$

which obviously includes quadratic terms. Similar to the attack we have discussed in Section 3.3.4, we use the following conservative approximation to reduce it to a first-order term:

$$z_k^e \varepsilon_k \approx B(\hat{z}^r) 2^{-(f_z+1)} \varepsilon_k. \tag{3.37}$$

As a result, the error $\hat{z}^e$ is turned into an affine form.

For affine operations, $x \pm y$, cx, and $x \pm c$, and non-affine operations, xy and x/y, we derive the corresponding AA-based floating-point computation models using the same methods as in the fixed-point error analysis, except that the new roundoff error term is replaced with the approximation in (3.37). We omit the detailed derivation, and present the results in the following.

- Addition $z = x \pm y$

$$\begin{aligned} \hat{z}^r &= \hat{x}^r \pm \hat{y}^r \\ \hat{z}^e &= \hat{x}^e \pm \hat{y}^e + B(\hat{z}^r)2^{-(f_z+1)}\varepsilon_k \end{aligned} \tag{3.38}$$

- Addition with a constant $z = x \pm c$

$$\begin{aligned} \hat{z}^r &= \hat{x}^r \pm c_0^r \\ \hat{z}^e &= \hat{x}^e \pm c_0^e + B(\hat{z}^r)2^{-(f_z+1)}\varepsilon_k \end{aligned} \tag{3.39}$$

- Multiplication with a constant $z = cx$

$$\begin{aligned} \hat{z}^r &= c_0^r \hat{x}^r \\ \hat{z}^e &= (c_0^r + c_0^e)\hat{x}^e + c_0^e \hat{x}^r + B(\hat{z}^r)2^{-(f_z+1)}\varepsilon_k \end{aligned} \tag{3.40}$$

- Multiplication $z = xy$

$$\begin{aligned} \hat{z}^r &= x_0^r y_0^r + \sum_{i=1}^{n} (y_0^r x_i^r + x_0^r y_i^r)\varepsilon_i + B(\sum_{i=1}^{n} x_i^r \varepsilon_i) B(\sum_{j=1}^{n} y_j^r \varepsilon_j)\varepsilon_t \\ \hat{z}^e &= B(\hat{x}^r)\hat{y}^e + B(\hat{y}^r)\hat{x}^e + B(\hat{z}^r)2^{-(f_z+1)}\varepsilon_k \end{aligned} \tag{3.41}$$

- Division $z = x/y$

$$\begin{aligned} \hat{z}^r &= \hat{f}(\frac{\hat{x}^r}{\hat{y}^r}) \\ \hat{z}^e &= B(\hat{f}(\frac{\hat{x}^r}{(\hat{y}^r)^2}))\hat{y}^e + B(\hat{f}(\frac{1}{\hat{y}^r}))\hat{x}^e + B(\hat{z}^r)2^{-(f_z+1)}\varepsilon_k. \end{aligned} \tag{3.42}$$

3.3.5 Experimental Results

In this section, we evaluate the accuracy and the speed of our static analysis approach, and compare it with other existing approaches, namely the IA-based approach and Monte Carlo simulation.

Although our complete solution includes both range and error analysis, we focus our evaluation on error analysis, since it is the main contribution of our work. However, the results from range analysis are provided for the benchmark test.

Methodology

Recall our goal of error analysis: estimate quickly and accurately the numerical errors that accrue given a candidate finite-precision data format. To evaluate the accuracy our error estimation, we compare it against the maximum roundoff error obtained by Monte Carlo simulation over a suitably large set of inputs (10^6 input samples are in our experiment). The maximum error we seek during simulation is the maximum difference between the finite-precision value and the "ideal" real value, which we take to be the IEEE double-precision version of the computation.

To statically estimate the error bound of a DSP algorithm, we have implemented the AA-based fixed-point and floating-point computation models in a C++ library. It overloads the common C++ arithmetic operators, which allows us to analyze a DSP program with minimal modification to the source code. A single execution of the modified code automatically computes an affine interval for each variable in th code and the relevant bounds.

To obtain the maximum error by Monte Carlo simulation, the very first step is to transform the original double-precision source code into to a custom-precision one. Despite the lack of support for fixed-point and custom floating-point by the standard C++, our simulation with finite-precision data format is made possible by parameterizable simulation libraries. For fixed-point, we choose a public library—SystemC [85], which allows users to specify any fixed-point data type, as long as the total bitwidth does not exceed 64 bits. In our fixed-point experiments, we assume 16 bits for the fractional part. Since the focus of the experiments is error analysis, we choose the maximum possible integer width, i.e., 48 bits, to assure no overflow occurs. For floating-point, we develop a custom floating-point library— *CMUfloat* [26], which supports exponent width from 1 to 8 bits and mantissa width from 1 to 23 bits. In all the floating-point experiments, we assume 16-bit mantissa and 8-bit exponent. With these custom libraries, we are able simulate two implementations of a DSP algorithm in parallel, one with finite-precision, and the other with the IEEE double-precision, and compare the outputs to obtained the roundoff error.

	AA error bound	IA error bound	Simulated max error
Fixed-point	0.00122	0.00337	0.00109
Floating-point	0.0431	0.0964	0.0203

Table 3.4: Comparison between AA-based and IA-based error analysis

Comparing AA to IA

To verify the advantage of error cancellation enabled by the novel affine modeling, we implement the conventional IA-based error analysis, discussed in Section 3.3.1, and compare the two approaches on a DSP application—the 8-input IDCT (Inverse Discrete Cosine Transform), which is widely used in image and video processing. In order to show the accuracy improvement gained by affine formulation, we use probabilistic bounding with confidence level $\lambda = 1$ in this experiment. The inputs of the IDCT are assumed to lie in the range of [-64, 64]. The results for both fixed-point and floating-point error analysis are shown in Table 3.4. The AA-based error bound is much tighter compared to the IA-based error bound. IA overestimates because it fails to consider the correlations among variables. We highlight an example of such correlations in the IDCT diagram in Figure 3.12. Since correlations on a data path are very common in DSP applications, our AA-based error models significantly improves accuracy compared to the IA-based models, with negligible computational cost.

From Table 3.4, we can also see that even though the AA error bound is better than the IA error bound, it is still not very tight, especially for floating-point implementation. In the next experiment, we will show how the error bound is further improved by probabilistic bounding.

Effects of probabilistic bounding

Although affine arithmetic improves the accuracy of error analysis by capturing correlations, the results may still be too pessimistic as it ignores distribution information. Probabilistic bounding incorporates distribution of an affine interval and estimates a confidence interval, instead of a hard bound. Moreover, it provides a flexible mechanism for users to specify the confidence level λ. Figure 3.13 shows the the probabilistic bounds with varying λ (abscissa) for a DSP algorithm—64-

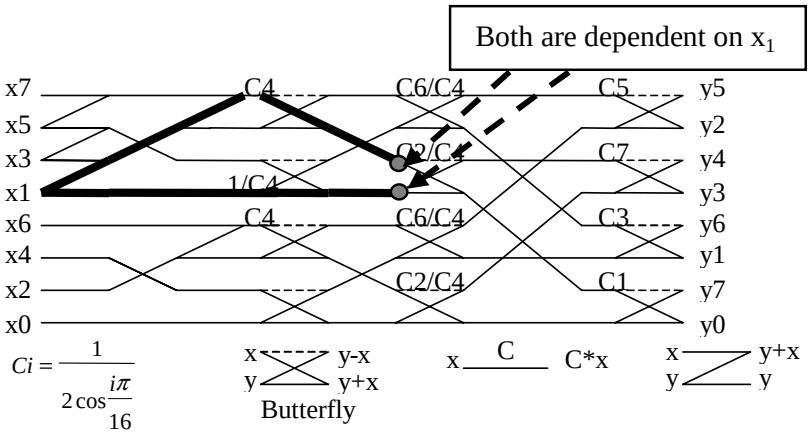


Figure 3.12: Data-flow of the IDCT algorithm

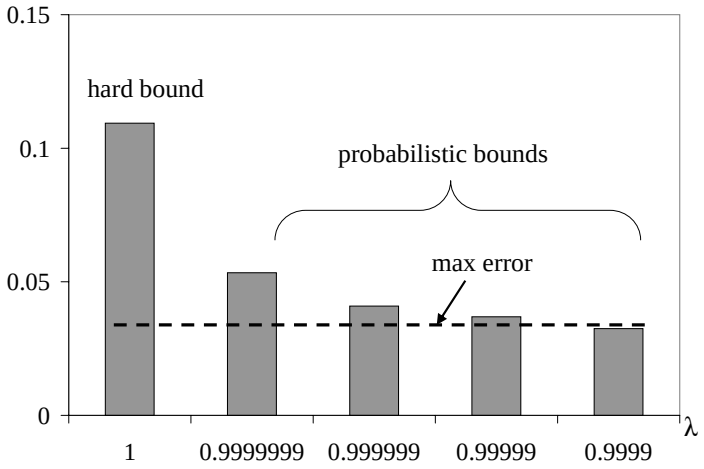


Figure 3.13: Probabilistic bounds for WHT64

input Walsh-Hadamard-Transform, or WHT64. The dashed line stands for the simulated maximum error. From left to right, we gradually decrease λ. The hard bound, with $\lambda = 1$, is too pessimistic, and hardly be useful in practice. If we slightly relax the confidence level λ, the probabilistic bound becomes much closer to the simulated maximum error.

Benchmark results

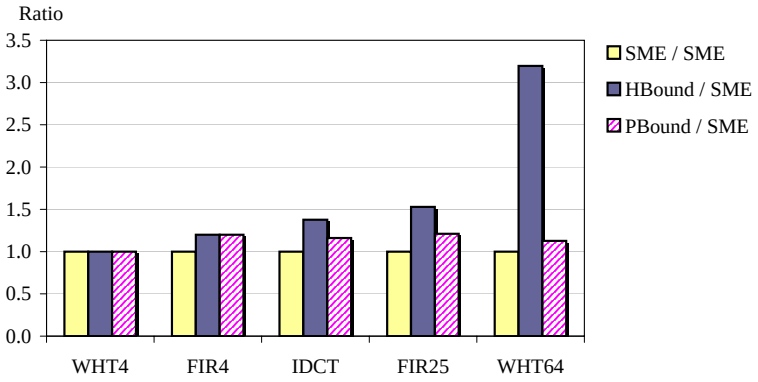
We test the applicability and accuracy of the proposed error models and the probabilistic bounding method on a variety of common DSP kernels, including WHT, FIR (Finite Impulse Response) filter, and IDCT. We assume the input range is [-64, 64] for all the kernels. We compare the sim-

ulated maximum error (SME), the hard error bound (HBound) and the probabilistic error bound with $\lambda = 0.9999$ (PBound) in Figure 3.14(a) and Figure 3.14(b), for fixed-point and floating-point, respectively. All the values in the figures are normalized with respect to the simulated maximum error. The x-axis is ordered by computational complexity, with WHT64 being the most complex one. For both fixed-point and floating-point error analysis, we have the following observations. First, the hard error bound is always greater than the maximum error. As computational complexity increases, this bound becomes looser (see WHT64). This verifies the asymptotic behavior suggested by the central limit theorem, i.e., as a random variable involves more uncertainties, the distribution gets closer to a normal distribution, and hence the mass is more concentrated on the center. Second, our probabilistic bounding method always offers a tight—yet reasonably accurate (with 99.9999% confidence)—bound to the maximum error, regardless of the computational complexity of the targeted DSP algorithm.

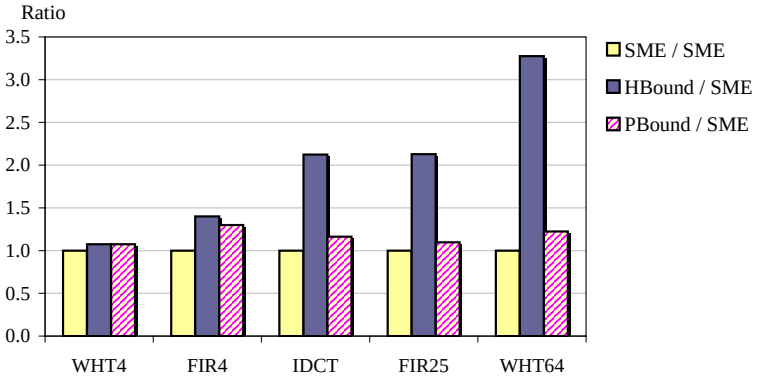
We also provide the results from range analysis in Figure 3.14(c), where the hard bound and the probabilistic bound are compared to the simulated maximum (SM). Note that the result is independent on the data format chosen. We find that that the probabilistic bound is even more accurate than that from error analysis, because less approximations are involved in range analysis.

Our static error analysis is attractive because of not only the accuracy, but also the computational efficiency. In contrast to Monte Carlo simulation, which requires hundreds of thousands of complete program executions, our method can predict error statically via a single program execution with overloaded operators. We compare the CPU times needed to compute the estimated error bound and the simulated maximum error on a 1.6GHz Pentium4 machine. Table 3.5 and 3.6 show the results for fixed-point and floating-point, respectively. Note that the CPU time required to compute the probabilistic bound is independent of λ. Although floating error analysis is slightly slower than its fixed-point counterpart, they both consistently achieve four to five orders of magnitude speedup compared to simulation of 10^6 input samples (see the fourth column).

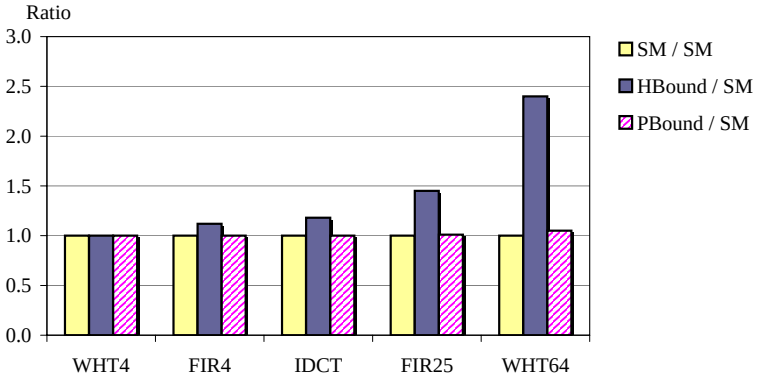
One may argue that smart simulation techniques may be employed so that much fewer input samples are needed to produce satisfactory results. To make an unbiased comparison, we calculate the number of samples that can be simulated within the CPU time needed for static analysis (see the fifth column). We can see that for fixed-point, less than 20 samples can be simulated within the



(a) Error analysis results for fixed-point



(b) Error analysis results for floating-point



(c) Range analysis results

Figure 3.14: Accuracy of error and range analysis

* WHT n : n-input WHT
* IDCT : 8-input IDCT
* FIR n : n-tap FIR filter
* SME : Simulated maximum error
* HBound : Hard bound
* PBound : Probabilistic bound
* SM : Simulated maximum

same amount of time, and for floating-point, about 30–100 samples can be simulated. No matter what simulation techniques are used, less than 100 samples are absolutely insufficient to produce reliable results. Therefore, our static error analysis is always a more efficient approach.

	CPU time (sec) for static analysis	CPU time (sec) for simulation	Speedup	Equivalent number of samples simulated
WHT4	3×10^{-5}	29	9.4×10^5	1
WHT64	1.5×10^{-3}	1147	7.5×10^5	1
FIR4	9.5×10^{-4}	50	5.3×10^4	19
FIR25	2.6×10^{-3}	356	1.4×10^5	7
IDCT8	1.1×10^{-3}	155	1.4×10^5	7

Table 3.5: Comparison of CPU time (fixed-point)

	CPU time (sec) for static analysis	CPU time (sec) for simulation	Speedup	Equivalent number of samples simulated
WHT4	0.002	31	1.6×10^4	64
WHT64	0.173	2854	1.7×10^4	60
FIR4	0.005	54	1.1×10^4	92
FIR25	0.018	331	1.8×10^4	54
IDCT8	0.01	282	2.8×10^4	35

Table 3.6: Comparison of CPU time (floating-point)

3.3.6 Demonstration Applications

In this section, we demonstrate how our AA-based error analysis can be used to assist finite-precision DSP design. The first application is an 8×8 DCT design in video encoding, we especially demonstrate how the estimation accuracy is improved by careful input correlation modeling. The second application is an IIR filter, through which we show that the AA-based error analysis is applicable not just to feed forward, or loop-free, systems, but also to feedback systems.

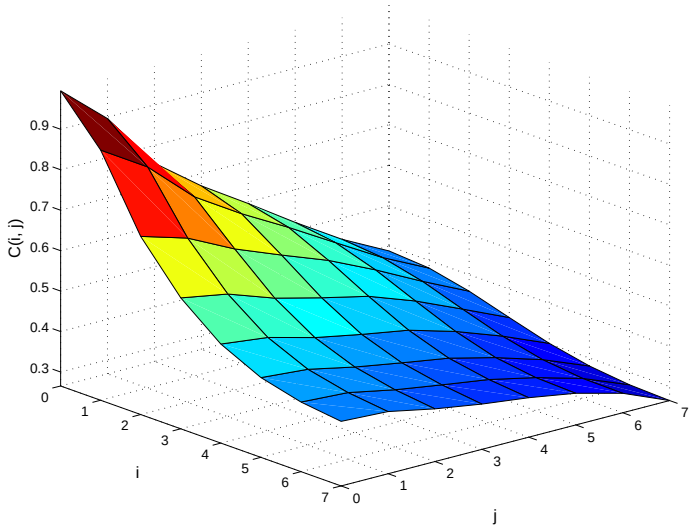


Figure 3.15: Correlation of pixels

$C(i, j)$ is the correlation coefficient of two pixels whose horizontal distance is i and vertical distance is j.

DCT in video encoding

When DCT (Discrete Cosine Transform) is used in image/video encoding, its inputs—the image pixels—are usually correlated based on their spatial relationship: the pixels that are close tend to have strong positive correlations. To visually show the correlation relationship, we generate a correlation coefficient plot for a 10 second video (one frame of the video is shown in Figure 3.7 in Section 3.3.1). For two pixels whose horizontal distance is i and vertical distance is j (i and j are non-negative), we denote by $C(i, j)$ their correlation coefficient. If they are perfectly correlated, $C(i, j) = 1$, and if they are uncorrelated, $C(i, j) = 0$. Figure 3.15 shows $C(i, j)$ for $0 \leq i, j \leq 7$. We can see that the correlation coefficient is close to 1 in the proximity of $(i = 0, j = 0)$, and gradually diminishes as i and j increase.

In presence of such input correlation patterns, we need to be cautious when performing static error analysis for custom precision DCT design. If the input correlation is not properly taken into account, it may lead to unwanted underestimation or overestimation. In this demonstration application, we employ PCA to model correlations among the DCT inputs. The goal is to search for the minimal mantissa width using the AA-based static error analysis.

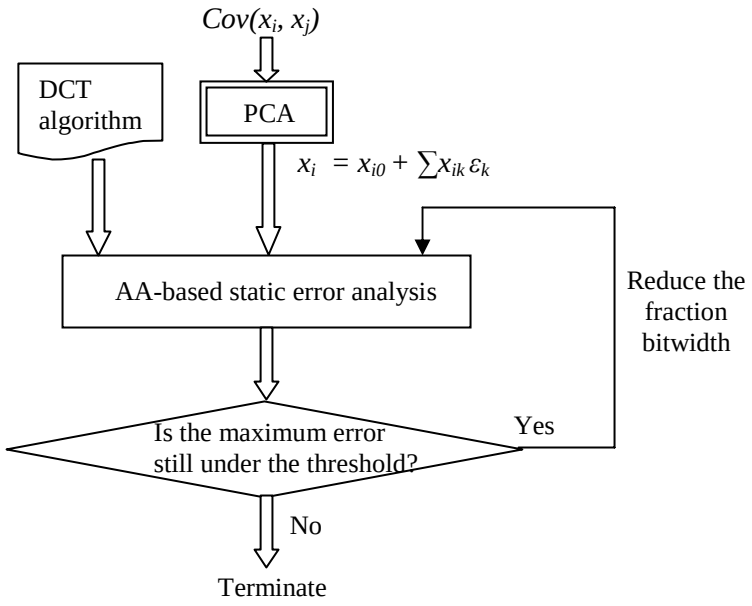


Figure 3.16: DCT design procedure

The design procedure is depicted in Figure 3.16. The first step is to model the program inputs as affine intervals. Normally, the DCT inputs can be modeled by $x_i = x_{i0} + x_{i1}\varepsilon_i$, where x_{i0} indicates the central value, and x_{i1} is the radius of the input distribution. However, by choosing a unique ε_i for each input, it is assumed that the inputs are independent. In our case, the inputs are strongly correlated, and hence are better off being modeled by a set of *correlated* affine intervals through PCA. Each input is then represented by a linear combination of a set of independent component ε_i's as

$$x_i = x_{i0} + \sum_{k=1}^{N} x_{ik}\varepsilon_k,$$

where the number of independent components, N, is less or equal to the number of inputs. For 8×8 2D-DCT, there are 64 inputs, and we keep the first 42 components because the rest are less than 0.1% of the largest component. It is worth mentioning that the purpose of using PCA here is not to reduce dimensionality, but to orthogonalize the inputs, and therefore it is unfavorable to apply aggressive component pruning. The second step is an iterative process of error analysis and bitwidth adjustment. Starting from an initial large bitwidth, we check the estimated maximum error against a certain threshold and progressively reduce the bitwidth until the accuracy requirement is violated. In our DCT design, we specify the threshold of the maximum output error to be 1, because

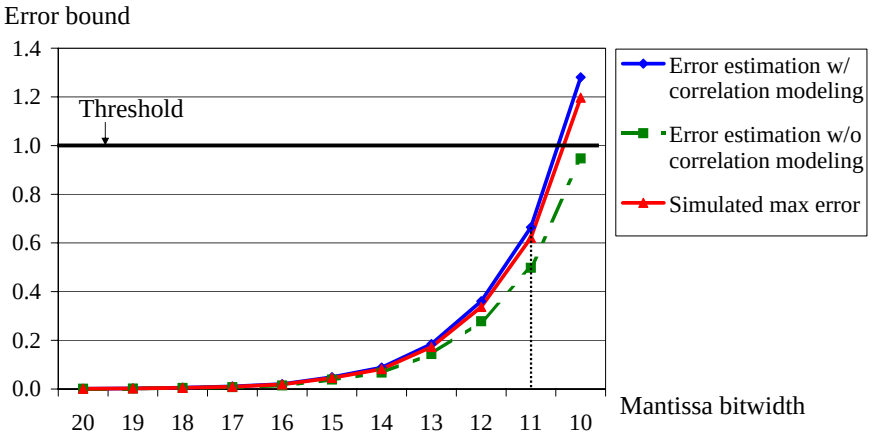


Figure 3.17: Error bound vs. mantissa width in DCT design

the output of DCT will eventually be rounded to the nearest integer.

The results of such iterative bitwidth reduction process are presented in Figure 3.17. The curves show how the maximum error changes while the mantissa bitwidth is decreased. We compare the results from static error analysis with and without input correlation modeling. As one would expect, considering input correlation provides more accurate error estimation, while assuming independent inputs yields error underestimation. From the curve, we can determine that the minimum mantissa bitwidth that satisfies the accuracy requirement is 11.

A feedback system—IIR filter

In a feed forward DSP system, the output always depends on a finite number of inputs and operations. As a contrast, in a feedback system, the output at the current time step depends not only on the inputs, but also the outputs from previous time steps. Since the AA-based error models preserve all the uncertainty terms that are related to the output, as more iterations are analyzed, the number of uncertainty terms definitely increases. This raises an important question for our error analysis: will the estimated output error keep growing endlessly and lose track of the actual error?

To investigate this, we conduct fixed-point error analysis on a common DSP feedback system—a second order IIR (Infinite Impulse Response) filter (Figure 3.18), whose transfer function is specified by

$$H(z) = \frac{1 + a_1 z^{-1} + a_2 z^{-2}}{1 - b_1 z^{-1} - b_2 z^{-2}}, \tag{3.43}$$

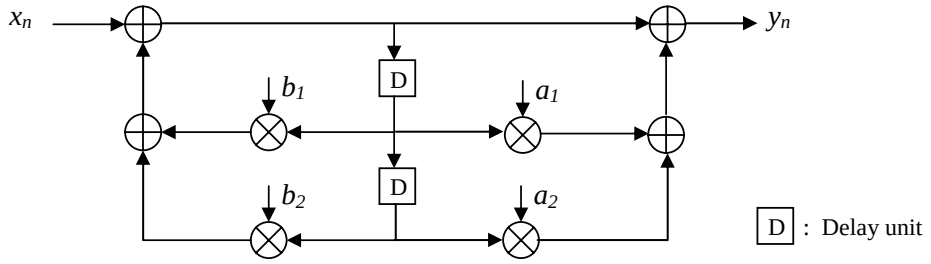


Figure 3.18: An example of feedback systems—a second order IIR filter

Iteration	5	10	15	20	25	30
Number of uncertainty terms	17	37	57	77	97	107

Table 3.7: Constant growth of uncertainty terms

where $\{a_1, a_2, b_1, b_2\} = \{0.355407,\ 1,\ 1.66664,\ 0.75504\}$. The values of b_1 and b_2 are chosen to ensure that the poles of the feedback system are within the unit circle, or in other words, the IIR filter is a stable system. The inputs are known to be independently distributed in [-64, 64]. Our objective is to find the minimal fraction width that guarantees the maximum error of the output is less than 0.1.

The first step is to check whether our static error analysis is applicable to feedback systems. We observe that as more iterations are analyzed, the number of uncertainty terms in the output grows constantly (see Table 3.7). What is more interesting is that the estimated output error does not grow at a constant speed. Figure 3.19 shows the error bound growth when the fraction bitwidth is fixed at 16. In the first a few iterations, it increases rapidly, and then the error growth slows down. After 25 iterations or so, it converges to a constant which is 0.019. We also run Monte Carlo simulation for 10^6 iterations, with a random input for each iteration. The maximum error encountered during simulation is 0.0182, which is tighted bounded by the estimated error bound 0.019. error.

The reason for the convergence is that *for a stable feedback system, the roundoff error itself is also stable*. By applying the AA-based error models to the second order IIR filter in (3.43), we get

$$E_n - b_1 E_{n-1} - b_2 E_{n-2} = \Phi_n + a_1 \Phi_{n-1} + a_2 \Phi_{n-2} \tag{3.44}$$

where E_i is the estimated output error at the ith iteration, and Φ_i is a sum of the input error and the roundoff errors introduced by the multiplications and the additions during the ith iteration. If we

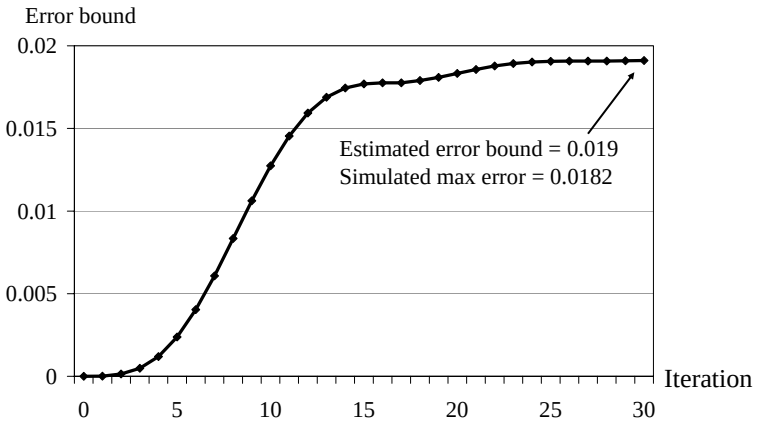


Figure 3.19: Convergence of error estimation in the IIR filter design

The estimated error bound increases rapidly in the first a few iterations, and then gradually converges to a constant which is 0.019. We also run Monte Carlo simulation for 10^6 iterations, with a random input for each iteration. The maximum error encountered during simulation is 0.0182, which is tighted bounded by the estimated error bound 0.019.

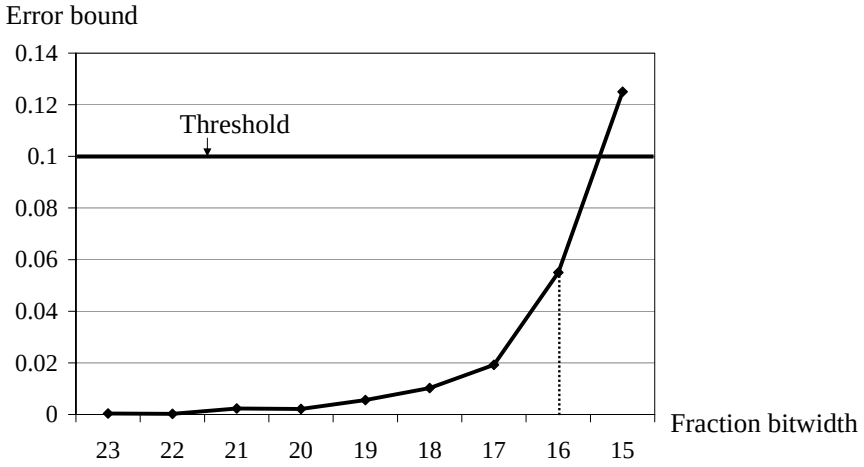


Figure 3.20: Error bound vs. Fraction width in IIR filter design

view Φ as the system input and E as the output, then (3.44) has exactly the same transfer function as the IIR filter in (3.43). Thus, E is also stable, and does not grow to infinity. Therefore, error analysis only needs to be conducted until the error bound converges. .

The second step is to search for the minimal fraction width, given an accuracy requirement. It is an iterative procedure, similar to what we have described for the DCT design. For each bitwidth choice, we run the IIR filter for 25 time steps to obtain the asymptotic error bound. It is then compared against the threshold 0.1. Figure 3.20 shows how the error bound increases as we reduce the bitwidth. To ensure the error is under 0.1, we choose 16 bits for the fraction part.

DCT algorithm selection

We illustrate here how our error analysis tool assists with fast design space exploration in the dimension of numerical precision. For a given algorithm, various implementations may lead to differerent numerical precisions due to different data paths, even with the same floating-point format. In this example, we consider a much more complicated kernel, a DCT type IV of size 64, which requires about 830 arithmetic operations on the data path. We generate four different implementations, based on algebraically different algorithms, using the DSP transform code generator SPIRAL [75], and compare the obtained error bounds. In this experiment, we specify λ to be 0.9999, which offers a confidence of $\lambda^c = 0.92$. The choice of λ does not affect the relative order of the error bounds for the four different algorithms.

In Figure 3.21, both the probabilistic error bound and the maximum error yield the same order for the four algorithms with respect to numerical accuracy: DCT4 $<$ DCT2 $<$ DCT3 $<$ DCT1, while the probabilistic error bound estimation is about 5000 times faster than running the simulation of one million random inputs. Note that the large spread in accuracy makes the choice of the algorithm a mandatory task.

3.4 Summary

The special representation form of affine arithmetic empowers it to efficiently handle correlations, but at the same time, brings pessimism to the bounds, especially for large scale problems. In this

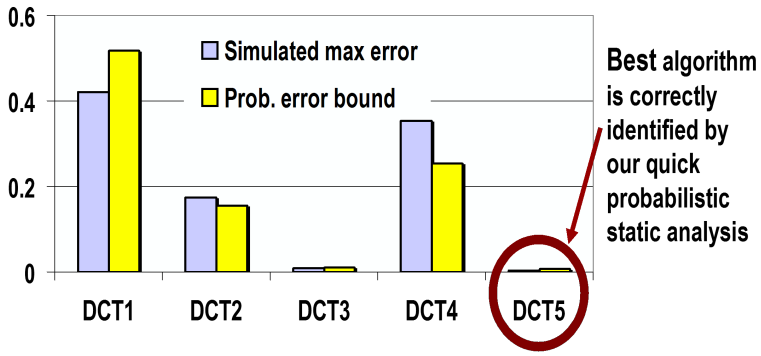


Figure 3.21: Comparison of four DCT algorithms

chapter, we alleviate this problem by using a novel probabilistic bounding method. We first provide a probabilistic interpretation for an affine interval, based on the Lindeberg-Feller Central Limit Theory, and then extend it to a 2D affine interval (a joint interval of two affine forms). These interpretations not only improve the accuracy of the bounds, but also help reduce pessimism in interval computations. In addition, we have discussed the means to initialize affine intervals from given probabilistic information. This provides a systematic approach for modeling correlations among the inputs of an interval algorithm, and further improve the accuracy of interval-based analysis.

We apply the improved affine arithmetic to a DSP application—range and error analysis, an essential step in finite-precision DSP design. Through common DSP algorithms, we have demonstrated the following:

1. AA is significantly better than IA in terms of the tightness of the bounds, since it handles correlations in DSP algorithms;

2. By using the probabilistic bounding method, the new interval technique offers bound estimates comparable to statistical simulation, with orders of magnitude speedup. Further, the accuracy does not degrade as the application complexity increases;

3. Accurate modeling of input correlations helps to improve the accuracy of interval-based analysis, as shown in the IDCT experiment.

One of the reasons that affine arithmetic performs so well on DSP applications is that they are mainly composed on linear functions. However, for nonlinear functions, such as multiplication and division, AA modeling employs lots of approximations. Therefore, the AA-based range and

error analysis is less suitable for nonlinear applications, especially for the ones with a long chain of exclusively non-affine operations. This limitation of the current AA will be treated in the next chapter.

Chapter 4

Asymmetric Probabilistic Bounding for Interval Operations

So far our applications of affine arithmetic are mainly on linear DSP transformations. What hinders its application to a wider range of problems is the undue pessimism in nonlinear interval computations. This brings us to a fundamental limitation of affine arithmetic, i.e., the center-symmetry of the interval implied by the affine form representation. Very often, the interval for the output of a nonlinear function is not symmetric around the center, and therefore pessimistic approximations have to be made in order to represent the output in a symmetric affine form.

To empower affine arithmetic to capture asymmetric intervals, and ultimately to improve the accuracy of nonlinear interval operations, we propose to enforce *asymmetric bounds* on an affine interval. This seemingly minor augmentation not only allows more information carried with basic affine intervals, but also brings opportunities for better algorithms for nonlinear interval operations.

In this chapter, we first elaborate on the pessimism of the existing algorithms for interval operations. Then, we define a new *asymmetric affine interval*, and discuss in great detail how we improve the accuracy of non-affine operations through enforcing asymmetric bounds and some other techniques, such as the *minivolume* approximation. Finally, we evaluate the improvements by applying the new affine interval analysis to nonlinear applications.

4.1 Motivation

We have seen in the previous chapter that affine arithmetic performs well on many DSP applications which involve mainly linear arithmetic operations. However, it becomes less accurate in very nonlinear applications. For example, the Cholesky decomposition [83]is dominated by non-affine functions, such as multiplication, division and square root. In order to apply affine arithmetic to a wider range of applications, the accuracy for non-affine interval operations has to be improved.

A fundamental limitation of affine arithmetic is the center-symmetry of affine intervals. According to the definition, in an affine interval $\hat{x} = x_0 + \sum_{i=1}^{N} x_i\varepsilon_i$, each noise symbol is distributed in [-1, 1], and hence the overall interval is symmetric to the center x_0. For a pair of affine intervals $(\hat{x}, \hat{y})$, the joint range is a polygon symmetric to the center (x_0, y_0), as we have discussed in Section 3.2.2. The computation models in affine arithmetic guarantee that the output of any computation is also a center-symmetric affine interval. This property highly limits the accuracy of interval computations. It is not difficult to see that for non-affine arithmetic operations, such as exp, $\times$, /, ..., etc., if the input intervals are center-symmetric, the true output interval may not be symmetric around the center any more, and the output interval obtained from the interval computation has to sacrifice accuracy in order to preserve center-symmetry.

Next, we discuss in more detail the particular problem associated with each type of interval operation.

4.1.1 The overshoot problem in unary non-affine functions

Unary non-affine functions, such as $exp(x)$, $log(x)$, $sqrt(x)$ and $1/x$, have a common *overshoot* problem: the bounds for the computed interval are always beyond the true bounds of the result. In other words, it always yields a pessimistic estimate. We illustrate this problem in Figure 4.1, using the exp function as an example. The input is $\hat{x} = x_0 + x_1\varepsilon$, and the output is computed according to the *Chebyshev approximation* [22]. In the figure, the output affine interval lies in the region bounded by the two solid straight lines. The range of this interval is center-symmetric, indicated by the bar on the left. However, the true range, indicated by the bar on the right, is not center-symmetric. Therefore, the Chebyshev approximation results in a pessimistic lower bound. In

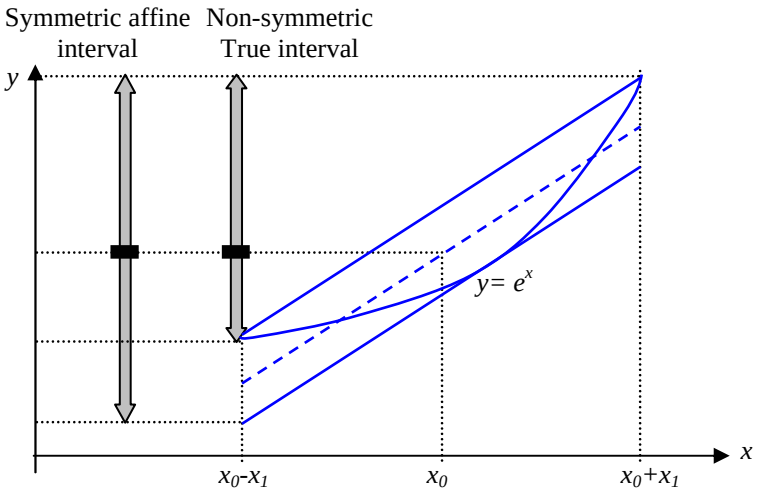


Figure 4.1: The exp function on affine interval

The exp function of an affine interval is performed using the Chebyshev approximation. The computed affine interval is center-symmetric, represented by the bar on the left, while the true interval is not center-symmetric, represented by the bar on the right.

general, for convex unary functions, such as $exp(x)$ and $1/x$, the lower bound is pessimistic, while for concave unary functions, such as $sqrt(x)$ and $log(x)$, the upper bound is pessimistic.

Note that all our discussions about unary functions in this chapter are limited to monotone functions. For a non-monotone function, interval extension is valid only when the input interval is within the monotone segment of the function. Trigonometric functions, such as $cos(x)$ and $sin(x)$, are common non-monotone functions. When the input interval is within the monotone segment, the interval extension of a trigonometric function can be derived using series expansion. Detailed analysis of trigonometric functions is beyond the scope of this thesis.

4.1.2 The pessimism in multiplication

When two affine intervals are multiplied, the result is not a linear combination of ε_i's any more. The inaccuracy in the interval multiplication is caused by approximating the high order terms to linear terms. The existing algorithm uses a fast, but very pessimistic approximation scheme, known as the *trivial range estimation* [22]. Given two inputs $\hat{x} = x_0 + \sum_{i=1}^{N} x_i\varepsilon_i$ and $\hat{y} = y_0 + \sum_{i=1}^{N} y_i\varepsilon_i$, we

compute the product as

$$\begin{aligned}\hat{x}\cdot\hat{y} &= x_0y_0 + \sum_{i=1}^{N}(y_0\cdot x_i + x_0\cdot y_i)\varepsilon_i + (\sum_{i=1}^{N} x_i\varepsilon_i)(\sum_{i=1}^{N} y_i\varepsilon_i) \\ &\approx x_0y_0 + \sum_{i=1}^{N}(y_0\cdot x_i + x_0\cdot y_i)\varepsilon_i + (\sum_{i=1}^{N} |x_i|)(\sum_{i=1}^{N} |y_i|)\varepsilon_k,\end{aligned} \tag{4.1}$$

where ε_k is a new noise symbol. This scheme usually overestimates the range of the high order terms. In the worst case, the overestimated range is four times the true range.

Here, we use a simple example to explain the overestimation. We have two correlated affine intervals

$$\hat{x} = 0 + \varepsilon_1 + \varepsilon_2$$
$$\hat{y} = 0 + \varepsilon_1 - \varepsilon_2.$$

Their joint range is shown in Figure 4.2(a). The product xy varies from -1 to 1 in this range, and the extremums are reached at the four points shown in the figure. However, if we perform the interval computation using trivial range estimation, the resulting interval for the product is

$$\begin{aligned}\hat{x}\cdot\hat{y} &= (\varepsilon_1 + \varepsilon_2)(\varepsilon_1 - \varepsilon_2) \\ &\approx 4\ \varepsilon_{new}.\end{aligned}$$

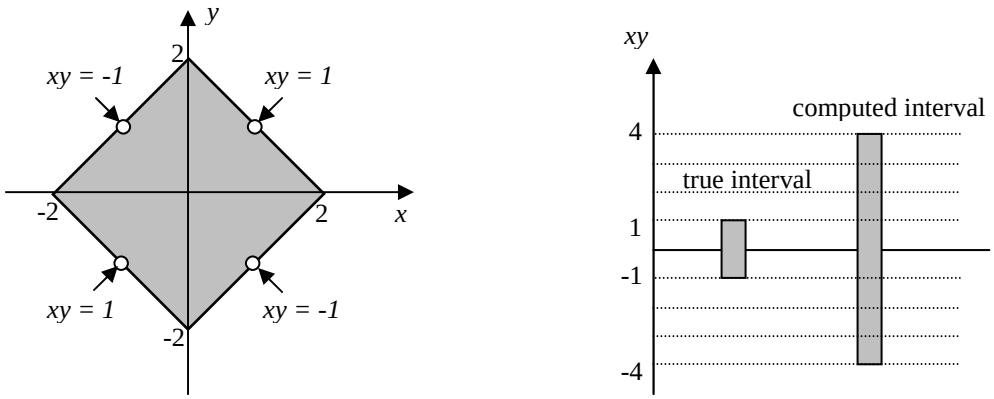
So the range for the computed interval is from -4 to 4, which is four times the true range (see Figure 4.2(b)).

4.1.3 The pessimism in division

Division on affine intervals is even more pessimistic than multiplication. First, the division $\hat{z} = \hat{x}/\hat{y}$ is computed indirectly from multiplication by

$$\hat{x}/\hat{y} = 1/\hat{y}\cdot\hat{x},$$

and therefore the pessimism in division is twofold, from both the multiplication and the reciprocal function. The former has the overestimation problem from the trivial range estimation, and the latter has the overshoot problem. Second, the distribution of the quotient x/y is often highly concentrated



(a) Joint range of x and y (b) Comparison of the true interval and the computed interval

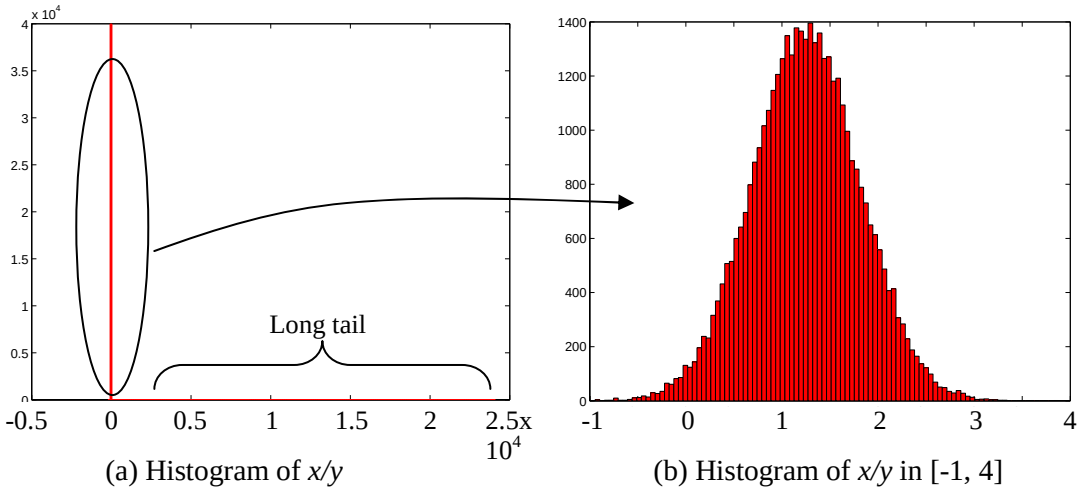
Figure 4.2: Overestimation in multiplication

In this example, $\hat{x} = 0 + \varepsilon_1 + \varepsilon_2$ and $\hat{y} = 0 + \varepsilon_1 - \varepsilon_2$. The computed range for xy is four times the true range.

and skewed to one side, especially when x and y are correlated and the noise symbols in the affine interval representations are normally distributed. A center-symmetric interval is very inaccurate to represent this type of distribution. We use the following example to demonstrate. Suppose

$$\hat{x} = 100 + 25\varepsilon_1 + 25\varepsilon_2$$
$$\hat{y} = 100 + 25\varepsilon_1 - 25\varepsilon_2,$$

where ε_1 and ε_2 are normally distributed. We randomly sample ε_1 and ε_2 from normal distributions and plot the histogram of x/y in Figure 4.3. It clearly shows that the majority of the distribution is around 1, while the tail of the distribution extends beyond 2.5×10^4. Therefore, the existing division algorithm, i.e., to have a center-symmetric interval to cover the whole range of x/y, is extremely pessimistic. In this case, a more sensible answer is to let $\hat{z}$ be in the range $[-1, 4]$, as suggested by Figure 4.3(b).



(a) Histogram of *x/y* (b) Histogram of *x/y* in [-1, 4]

Figure 4.3: Histogram of x/y

In this example,$\hat{x} = 100 + 25\varepsilon_1 + 25\varepsilon_2$ and $\hat{y} = 100 + 25\varepsilon_1 - 25\varepsilon_2$, where ε_1 and ε_2 are normally distributed. The histogram of x/y is highly concentrated in [-1, 4], and has a long tail extending beyond 2.5×10^4.

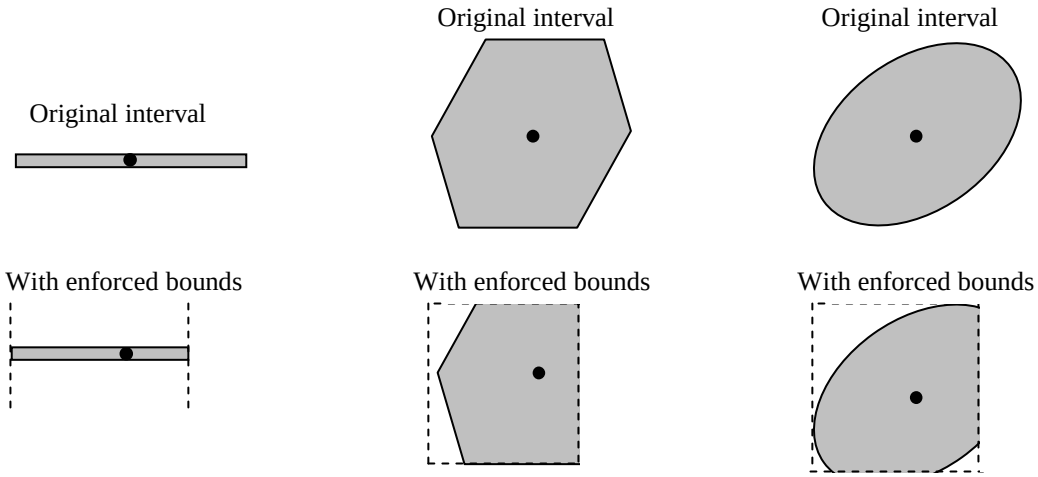
4.2 Core Theories

4.2.1 Enforcing asymmetric bounds

Definition

We introduce an enhanced representation for affine interval—*asymmetric affine interval*—to mitigate the symmetry problem, and hence to improve the accuracy of non-affine operations. It consists of a basic affine interval and two *enforced bounds*, asymmetric around the central point. Please note that a basic affine interval also has two *implied probabilistic bounds*, $\underline{x}_\lambda$ and $\overline{x}_\lambda$, which are symmetric around the center and bound the true range with a certain probability. To distinguish from the implied symmetric bounds, we denote the enforced asymmetric bounds by x_l (for the lower bound) and x_h (for the upper bound). So an asymmetric affine interval is represented by

$$\begin{aligned} &\hat{x}^a = \{x_0 + \sum_{i=1}^{N} x_i\varepsilon_i \ , \ x_l \ , \ x_h\} \\ &x_l \geq \underline{x}_\lambda \\ &x_h \leq \overline{x}_\lambda \end{aligned} \tag{4.2}$$



(a) 1D affine interval (b) 2D affine interval (c) 2D affine interval with probabilistic interpretation

Figure 4.4: Representing asymmetric region with asymmetric affine intervals

The condition states that the enforced lower bound x_l is effective only when it is larger than the probabilistic lower bound $\underline{x}_\lambda$, and the enforced upper bound x_h is effective only when it is smaller than the probabilistic upper bound $\overline{x}_\lambda$. In the rest of this chapter, we also use a simplified notation of (4.2), i.e. $\hat{x}^a = \{\hat{x}, [x]\}$, where $[x]$ stands for the enforced bounds, i.e., $[x] = [x_l, x_h]$.

With enforced bounds, a new affine interval $\hat{x}^a$ can represent a non-center-symmetric range. We show some examples in Figure 4.4. The first example is for a 1D affine interval (figure (a)), where the line segment described by the original interval is limited to be center-symmetric, while the new interval is capable of representing a line segment that is asymmetric around the center. The next two examples are for a 2D affine interval $\hat{x}^a \times \hat{y}^a$, in the cases of few noise symbols (figure (b)) and many noise symbols (figure (c)), respectively. Its boundary is originally restricted to be a center-symmetric polygon or an ellipse, but now is extended to be an asymmetric part of a polygon or ellipse that is intersected with a bounding box.

Asymmetric bounds are especially powerful in curing the overshoot problem. A quick demonstration is the following. Suppose $y = exp(x)$, where x is in $[a, b]$. As we have shown previously, the resulting affine interval for y implies an overly pessimistic lower bound. This problem can be fixed simply by adding a lower bound $y_l = exp(a)$. However, in general, computing with asymmet-

ric affine intervals is more complicated than this.

Interval operations with enforced bounds

An asymmetric affine interval can be viewed as a combination of a basic affine interval $\hat{x}$ and an enforced bound $[x]$. However, an interval operation on asymmetric affine intervals is usually more than just performing computations on the two parts separately and then aggregating the answers at the end. The coexistence of the affine interval and its asymmetric bounds affects each step during the interval computation. The only exceptions are the perfectly affine functions, namely, $x \pm y$, cx, and $x \pm c$, where the inputs having asymmetric bounds does not affect the interval computation itself, but rather passes the asymmetric bounds to the output. So in the rest of this chapter, we focus our discuss on the non-affine operations.

More specifically, an interval operation on asymmetric affine intervals is defined as $\{\hat{z}, [z]\} \leftarrow \{\hat{X}, [X]\}$, where upper case suggests that the input maybe multidimensional. The goal is to compute $\hat{z}$ and $[z]$ as accurately as possible, based on the joint information of $\hat{X}$ and $[X]$. There are two main steps in the computation:

- Step 1: Computing the interval. This corresponds to the procedure of $\hat{z} \leftarrow \{\hat{X}, [X]\}$. Because the selection of an affine function that approximates the target function is highly dependent on the input range, both the affine form $\hat{X}$ and the enforced bounds of the input $[X]$ have to be taken into account when considering the input range.

- Step 2: Computing the bounds. This step corresponds to the procedure of $[z] \leftarrow \{\hat{X}, [X]\}$. The enforced bounds for the output depend not only on the inputs' enforced bounds $[X]$, but also on the affine interval $\hat{X}$, since the latter carries important information on the input correlation.

In the next two sections, we will discuss in more detail unary and binary non-affine interval operations, evolving the two steps outlined above.

4.2.2 Unary operations on an asymmetric affine interval

Common unary non-affine operations include $z = exp(x)$, $z = log(x)$, $z = sqrt(x)$ and $z = 1/x$. The corresponding interval computation is, given the target function $z = f(x)$ and the asymmetric affine interval $\hat{x}^a = \{\hat{x}, [x]\}$, to evaluate the affine form $\hat{z}$ as well as the enforced bounds $[z]$.

Recall that the key in finding $\hat{z}$ is to decide an affine function $f^*(x) = ax + b$ that approximates the non-affine function $f(x)$. The selection of the approximating function is affected by the input range U^x. When x does not have enforced bounds, the affine form of x implies two symmetric probabilistic bounds, i.e., $U^x = [\underline{x}_\lambda, \overline{x}_\lambda]$, as we have presented in the previous chapter. When x has enforced bounds, the input range U^x becomes $[x_l, x_h]$, which is no wider than $[\underline{x}_\lambda, \overline{x}_\lambda]$. Based on the new U^x, we can determine the new approximating function, and hence the resulting affine form $\hat{z}$. This affine form is a more accurate estimate for z, since the new input range $[x_l, x_h]$ is less pessimistic. However, the tradeoff, of course, is that this more accurate approximation will be more expensive to compute.

Another impact of enforcing asymmetric bounds is that it mitigates the overshoot problem. Although the minimal and maximal values of z implied by the affine form $\hat{z}$ may be too pessimistic, we can correct the lower and upper bounds by adding additional constraints $z_l = min(f(x_l), f(x_h))$ and $z_h = max(f(x_l), f(x_h))$. The abstract algorithm for unary functions is described in Figure 4.5.

Next, we use the example of $exp(x)$ to illustrate the impacts of asymmetric bounds. Figure 4.6 (a) shows the original interval computation without enforced bounds. The nonlinear curve represents the function $exp(x)$, and the space between the two parallel lines represent the resulting affine interval. We can see that the range reached by the affine interval (indicated by the bar in the figure) is symmetric to the center $y_0 = exp(x_0)$, and is too pessimistic at the lower end. As a comparison, Figure 4.6(b) shows the result by the new interval computation with enforced bounds. First, the two parallel lines become closer due to the change of the input range U^x. Second, the two enforced bounds of z are asymmetric to the center, and the lower bound $z_l = exp(x_l)$ corrects the overshoot problem of the affine interval.

Finally, we give the complete algorithm for the $\exp$ function in Figure 4.7. Other unary functions on an asymmetric affine interval are similar to the new exponential function, and the complete

Input: $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i,\ [x_l, x_h]$

Output: $\hat{y} = y_0 + \sum_{i=1}^{N} y_i \varepsilon_i + \varepsilon_{N+1},\ [y_l, y_h]$

$f\ (\hat{x}, x_l, x_h)\{$

$a = x_l$

$b = x_h$

$slope = \frac{f(b)-f(a)}{b-a}$

x equals the solution to $f'(x)$ = slope

$c_1 = f(x) - slope \cdot x$

$c_2 = f(b) - slope \cdot b$

$\hat{y} = \hat{x} \cdot slope + \frac{c_1+c_2}{2} + \frac{c_1-c_2}{2}\varepsilon_{N+1}$

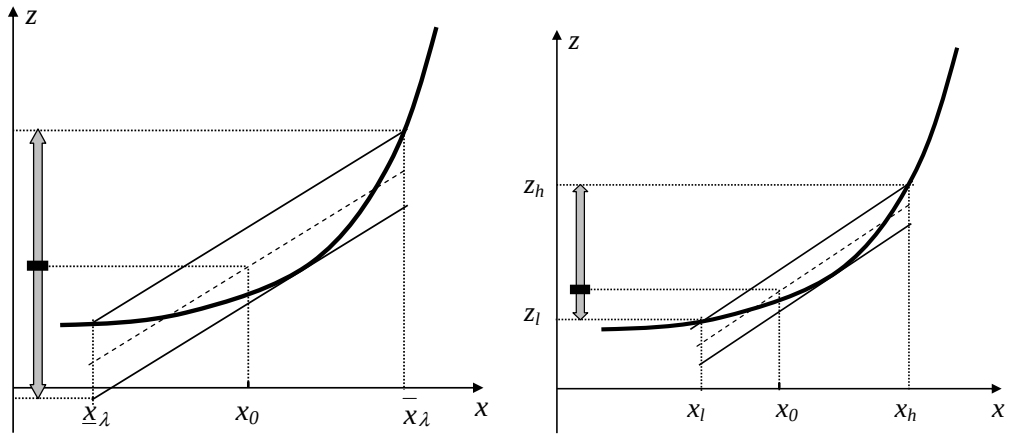
$y_l = min(f(a), f(b))$

$y_h = max(f(a), f(b))$

output $\{\hat{y}, y_l, y_h\}$

$\}$

Figure 4.5: Abstract algorithm for unary functions with asymmetric bounding



(a) The original exp function (b) The new exp function

Figure 4.6: Performing the exp function on an affine interval

Input: $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i,\ [x_l, x_h]$

Output: $\hat{y} = y_0 + \sum_{i=1}^{N} y_i \varepsilon_i + \varepsilon_{N+1},\ [y_l, y_h]$

exp ($\hat{x}$, x_l, x_h){

$a = x_l$

$b = x_h$

$slope = \frac{e^b - e^a}{b-a}$

$c_1 = slope \cdot (1 - \log(slope))$

$c_2 = e^b - b \cdot slope$

$\hat{y} = \hat{x} \cdot slope + \frac{c_1+c_2}{2} + \frac{c_1-c_2}{2}\varepsilon_{N+1}$

$y_l = e^a$

$y_h = e^b$

output $\{\hat{y}, y_l, y_h\}$

}

Figure 4.7: The $\exp$ function on an asymmetric affine interval

algorithms for the $\log$, reciprocal and sqrt functions are shown in Figure 4.8–4.10.

Input: $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i,\ [x_l, x_h]$

Output: $\hat{y} = y_0 + \sum_{i=1}^{N} y_i \varepsilon_i + \varepsilon_{N+1},\ [y_l, y_h]$

log ($\hat{x}$, x_l, x_h){

$a = x_l$

$b = x_h$

$slope = \frac{\log b - \log a}{b - a}$

$c_1 = \log(1/slope) - 1$

$c_2 = \log b - b \cdot slope$

$\hat{y} = \hat{x} \cdot slope + \frac{c_1 + c_2}{2} + \frac{c_1 - c_2}{2} \varepsilon_{N+1}$

$y_l = \log a$

$y_h = \log b$

output $\{\hat{y}, y_l, y_h\}$

}

Figure 4.8: The $\log$ function on an asymmetric affine interval

x_l has to be positive in order for the $\log$ function to be valid.

4.2.3 Binary operations on asymmetric affine intervals

As we have seen, unary operations on asymmetric affine intervals are fairly straightforward, as long as the input range is correctly identified. However, developing the two-step procedure for binary operations is far more challenging, especially for non-affine functions, such as multiplication and division. There are two major reasons for the arising difficulties. First, the joint range of the two inputs may fall into four different categories, namely, symmetric polygon, asymmetric polygon, full ellipse, and partial ellipse that is inside a bounding box. When the number of noise symbols in the inputs is small, the joint range is better described by a polygon. Otherwise, an approximating ellipse should be used to reduce pessimism as well as complexity. Therefore, the interval operation has to

Input: $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i,\ [x_l, x_h]$

Output: $\hat{y} = y_0 + \sum_{i=1}^{N} y_i \varepsilon_i + \varepsilon_{N+1},\ [y_l, y_h]$

reciprocal ($\hat{x}$, x_l, x_h){

$a = x_l$

$b = x_h$

$slope = -\frac{1}{ab}$

$c_1 = \frac{1}{a} + \frac{1}{b}$

$if(b > 0)$

$c_2 = \frac{2}{\sqrt{ab}}$

$else$

$c_2 = -\frac{2}{\sqrt{ab}}$

$\hat{y} = \hat{x} \cdot slope + \frac{c_1+c_2}{2} + \frac{c_1-c_2}{2}\varepsilon_{N+1}$

$y_l = \frac{1}{b}$

$y_h = \frac{1}{a}$

output $\{\hat{y}, y_l, y_h\}$

}

Figure 4.9: The reciprocal function on an asymmetric affine interval

The input range $[x_l, x_h]$ should not include zero in order for the reciprocal function to be valid.

Input: $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i,\ [x_l, x_h]$

Output: $\hat{y} = y_0 + \sum_{i=1}^{N} y_i \varepsilon_i + \varepsilon_{N+1},\ [y_l, y_h]$

sqrt ($\hat{x}$, x_l, x_h){

a = x_l

b = x_h

$slope = \frac{1}{\sqrt{a}+\sqrt{b}}$

$c_1 = \frac{\sqrt{a}+\sqrt{b}}{4}$

$c_2 = \frac{\sqrt{ab}}{\sqrt{a}+\sqrt{b}}$

$\hat{y} = \hat{x} \cdot slope + \frac{c_1+c_2}{2} + \frac{c_1-c_2}{2}\varepsilon_{N+1}$

$y_l = \sqrt{a}$

$y_h = \sqrt{b}$

output {$\hat{y}$, y_l, y_h}

}

Figure 4.10: The sqrt function on an asymmetric affine interval

x_l has to be non-negative in order for the sqrt function to be valid.

first identify the shape of the joint range, and then choose the algorithm that suits the particular category. Second, the original algorithms for multiplication and division on affine intervals described by Stolfi [22] provide very pessimistic results, and do not offer a mechanism to incorporate a more complex input range. Hence, it is difficult to extend the original algorithms to handle the enforced asymmetric bounds.

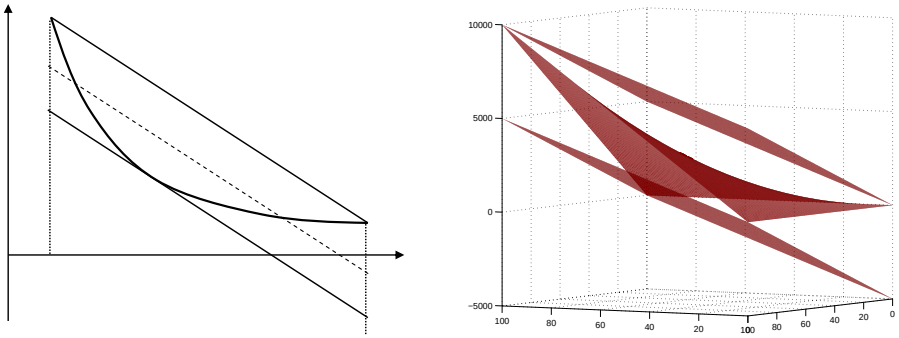
We propose a systematic approach, called the *minivolume approximation*, for the first step of the interval operation, i.e., to compute the interval $\hat{z}$ based on the joint range of the inputs. The key in this step is to choose an affine function f^* to approximate the target non-affine function f. The minivolume approximation provides a guideline on how to choose f^* so that a certain measure of the approximation error is minimized. The algorithm is also able to deal with any type of joint range of the inputs. In addition, even when the two inputs do not have enforced bounds, this algorithm is still superior to the original ones in that it provides tighter interval for the output.

The second step is to compute the enforced bounds for the output. A mathematical description of this procedure is to find the extreme values of $f(x, y)$ in the joint range of the two inputs. Better yet, we are interested in the "probabilistic" extreme values, i.e., the range that captures a high percentage of $f(x, y)$. However, the probabilistic asymmetric bounds are not always easy to compute. In cases where computing the "probabilistic" extreme values is infeasible or prohibitively expensive, we use the extreme values instead. Since this procedure is highly dependent on the specific function f, we will offer detailed discussions in the later sections dedicated to multiplication and division.

Minivolume approximation

In this part, we discuss in more detail the mechanics of the minivolume approximation. Recall in affine arithmetic discussed in [22] (see Section 2.2), a general rule for non-affine functions on affine intervals is the *Chebyshev approximation*, i.e., to minimize the maximum absolute error between the target function and its affine approximation. It is straightforward for unary functions, and the corresponding algorithm is also called the *minimax approximation* in [22]. Here, we develop the algorithm for the Chebyshev approximation for binary functions, and call it the *minivolume approximation*.

A geometric view of the minimax criteria helps to illustrate the connection between the two



(a) Minimax criteria (b) Minivolume criteria

Figure 4.11: Approximation criteria for non-affine functions

In (a), a unary function is represented by the nonlinear curve. The approximated affine form for the output is the space between the two parallel lines. Minimax criteria is to minimize the vertical distance between the two lines, while guaranteeing the nonlinear curve is bounded by the two lines over the range of x.
In (b), a binary function is represented by the nonlinear surface. The approximated affine form for the output is the space between the two parallel planes. The minivolume criteria is to minimize the vertical distance between the two planes, while guaranteeing the nonlinear surface is bounded by the two planes over the joint range of x and y.

algorithms. For a unary nonlinear interval function $f(\hat{x})$, we seek an affine form $= A\hat{x} + B + C\varepsilon$ to approximate the true result. In Figure 4.11(a), a nonlinear function is represented by a nonlinear curve, and the affine form $A\hat{x} + B + C\varepsilon$ describes the space between two parallel lines. Hence the minimax criteria is equivalent to minimizing the distance between the two lines that bound the nonlinear curve over the range of $[\underline{x}\ ,\ \overline{x}]$. This distance serves as a measure of the approximation error. Enlightened by this geometric interpretation, we develop a similar approximation criteria for binary functions. A nonlinear binary function is represented by a nonlinear surface in the 3D space (see Figure 4.11(b)). The resulting affine form $= A\hat{x} + B\hat{y} + C + D\varepsilon$ describes the space between two parallel planes, and the distance between the two planes indicates the maximum error between the approximating affine function and the original non-affine function. Using the similar principles as in the minimax criteria, we require these two planes to satisfy the following conditions:

1. The two planes should bound the nonlinear surface over the joint range of x and y
2. The vertical distance between the two planes should be minimized.

Since the volume between the two planes equals to the vertical distance times the area over the joint

range of x and y, minimizing the vertical distance is equivalent to minimizing the volume of the space between the two bounding planes. Therefore, we call these two conditions the *minivolume* approximation criteria.

The minivolume criteria gives a general guideline as to how to choose the approximating function. However, developing the solution based on the minivolume criteria is not a trivial task. Suppose a binary function is $z = f(x, y)$, and the joint range of x and y is U^{xy}. The key is to find the two bounding planes that satisfy the criteria. They are expressed as $z_1 = Ax + By + C_1$ and $z_2 = Ax + By + C_2$, where $C_1 > C_2$. Once the two bounding planes are found, the resulting affine interval is

$$\hat{z} = A\hat{x} + B\hat{y} + \frac{C_1 + C_2}{2} + \frac{C_1 - C_2}{2}\varepsilon_k \tag{4.3}$$

where ε_k is a new noise symbol, indicating the uncertainty of the approximation error. By letting $\varepsilon_k = \pm 1$, we can clearly see that this affine interval is bounded by z_1 and z_2.

Finding the two bounding planes can be formulated as a minimization problem:

$$\begin{aligned}
&\text{minimize } C_1 - C_2 \\
&\text{subject to} \\
&\qquad z \geq Ax + By + C_2 \\
&\qquad z \leq Ax + By + C_1 \\
&\qquad C_1 > C_2 \\
&\qquad x, y \in U^{xy}
\end{aligned}$$

where A, B, C_1 and C_2 are variables. This involves a time-consuming optimization procedure in order to determine all the unknown variables.

To simplify the problem, we fix A and B to be the partial derivatives at the center point (x_0, y_0):

$$\begin{aligned}
A &= \frac{\partial z}{\partial x}(x_0, y_0) \\
B &= \frac{\partial z}{\partial y}(x_0, y_0).
\end{aligned}$$

Further, since there are no constraints on the relationship of C_1 and C_2 other than $C_1 > C_2$, the optimization problem can be separated into two parts, i.e., minimizing C_1 and maximizing C_2.

Hence a suboptimal, but more tractable, formulation for the minivolume approximation is

$$\begin{aligned} &\text{minimize } C_1 \quad \text{and} \quad \text{maximize } C_2 \\ &\text{subject to} \\ &\qquad z \geq Ax + By + C_2 \\ &\qquad z \leq Ax + By + C_1 \\ &\qquad C_1 > C_2 \\ &\qquad x, y \in U^{xy} \end{aligned} \tag{4.4}$$

where only C_1 and C_2 are unknown, and $A = \frac{\partial z}{\partial x}(x_0, y_0), B = \frac{\partial z}{\partial y}(x_0, y_0)$.

The optimization problem in (4.4) is equivalent to finding the extreme values of the function $z - (Ax + By)$, which we call the *distance function*, over the range of U^{xy}. Therefore the solution for the plane parameters is the following

$$\begin{aligned} A &= \frac{\partial z}{\partial x}(x_0, y_0) \\ B &= \frac{\partial z}{\partial y}(x_0, y_0). \\ C_1 &= \max_{x,y \in U^{xy}} (z - (Ax + By)) \\ C_2 &= \min_{x,y \in U^{xy}} (z - (Ax + By)) \end{aligned} \tag{4.5}$$

The solution guarantees that over U^{xy}, the true result $f(x, y)$ falls in the range described by the affine interval $\hat{z}$, computed using (4.3) and (4.5). In addition, given the fixed plane parameters A and B, the volume between the two bounding planes of the affine interval $\hat{z}$ is minimized.

The main computational effort in the minivolume approximation is in finding the extreme values of the distance function $f(x, y) - (Ax + By)$ over the range of U^{xy}, which is dependent on the specific form of $f(x, y)$. The detailed procedures for multiplication and division are provided in the next two sections.

4.2.4 Multiplication on asymmetric affine intervals

Let us now consider the multiplication operator on asymmetric affine intervals, i.e., the evaluation of $z = xy$, given the asymmetric affine forms $\hat{x}^a = \{\hat{x}, [x]\}$ and $\hat{y}^a = \{\hat{y}, [y]\}$ for the operands x

and y. It consists of two major steps: one is to compute the affine interval $\hat{z}$ using the minivolume approximation discussed in the previous section, and the other is to compute the enforced bounds $[z]$, or to find the extreme values of xy in the joint range of x and y. In this section, we first talk about the general procedures for these two steps in the case of interval multiplication, and then separate the discussion based on whether the joint range is a polygon or an ellipse, and dive into more details for each case.

The main task in the minivolume approximation is to compute the parameters for the two bounding planes. In the case of multiplication, the parameters A and B are

$$A = \frac{\partial(x \cdot y)}{\partial x}(x_0, y_0) = y_0 \quad , \quad B = \frac{\partial(x \cdot y)}{\partial y}(x_0, y_0) = x_0,$$

and the distance function is

$$\begin{aligned} & D(x,y) = x \cdot y - (Ax + By) \\ =& (x_0 + \sum_i x_i\varepsilon_i) \cdot (y_0 + \sum_i y_i\varepsilon_i) - y_0 \cdot (x_0 + \sum_i x_i\varepsilon_i) - x_0 \cdot (y_0 + \sum_i y_i\varepsilon_i) \\ =& (\sum_i x_i\varepsilon_i) \cdot (\sum_i y_i\varepsilon_i) - x_0 y_0. \end{aligned} \tag{4.6}$$

The remaining two parameters, namely, C_1 and C_2, equal to the extreme values of the distance function over the joint range of x and y. Let

$$u = x - x_0 = (\sum_i x_i\varepsilon_i)$$
$$v = y - y_0 = (\sum_i y_i\varepsilon_i).$$

Then according to (4.6), the distance function reaches the extremes when uv are maximized or minimized, over the joint range of u and v. This joint range U^{uv} has the same shape as U^{xy}, but centers at $(0, 0)$. Its shape falls into one of the four categories, namely, symmetric polygon, non-symmetric polygon, full ellipse, and partial ellipse inside a bounding box. However, regardless of the shape of the joint range has, it is easy to prove that the extreme values of uv must be reached on the perimeter. Suppose (u_1, v_1) is a point that is not on the perimeter. Then on the perimeter, there must exist two points (u_1, v_2) and (u_1, v_3) such that $v_2 \leq v_1 \leq v_3$. The following relationship holds:

$$u_1 v_2 \leq u_1 v_1 \leq u_1 v_3, \text{ when } u_1 \geq 0,$$

and

$$u_1 v_3 \leq u_1 v_1 \leq u_1 v_2, \text{ when } u_1 < 0.$$

Hence, the extreme value of uv must be reached at a point on the perimeter. Therefore the parameters C_1 and C_2 are obtained by *tracing* the product uv along the perimeter of the joint range U^{uv}.

The second step is to find the enforced bounds for the output z. Ideally, we would like to compute the "probabilistic" extreme values of xy over the joint range U^{xy}, meaning the asymmetric bounds that capture a high percentage of all possible values of xy. However, we found it extremely difficult without any knowledge on the distribution of xy. Even if x and y are normally distributed, there is still no closed-form solution to the distribution of the product, due to their arbitrary correlation relationship. Often, people resort to a numerical integration method to seek the distribution of xy [63], which is too expensive to be adopted in interval computation. Therefore we use the extreme values of xy as the enforced bounds. Following the same argument as in the first step, we can prove that the extremums of xy must be on the perimeter of the joint range of x and y. So we compute the enforced bounds $[z]$ by tracing the product xy along the perimeter of U^{xy}.

Case A—U^{xy} is a polygon

When the number of noise symbols in $\hat{x}$ and $\hat{y}$ is small and the distributions of the noise symbols are not normal, it is better to describe the joint range U^{xy}, and accordingly, U^{uv}, as a polygon.[1] More precisely, when there exists enforced bounds for x and y, the polygon may be asymmetric. Otherwise, it is symmetric around the center point. A few possible shapes of U^{xy} are shown in Figure 4.12.

Now let's first trace along the perimeter of U^{uv} to solve the minivolume approximation problem. No matter what shape U^{uv} has, the polygon is composed of a limited number of connected line segments, which can be sorted by their slopes. On each line segment, the extreme values of uv are possible at three points: the two end points and one potential internal point. This is proved as the

[1] When the number of noise symbols is large or the distributions of the noise symbols are normal, the joint range can be regarded as an ellipse.

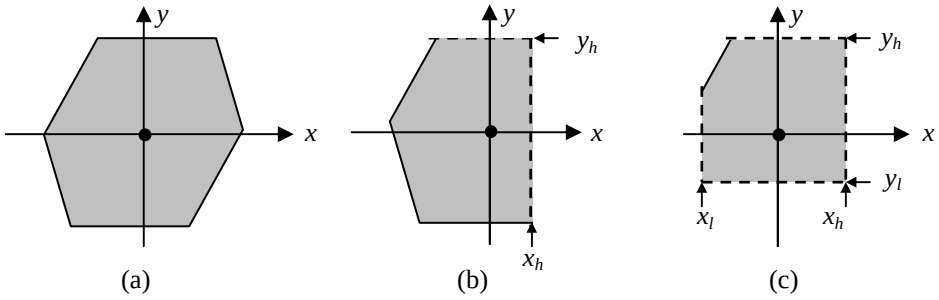


Figure 4.12: Examples of U^{xy}

following. We write each side of the polygon as

$$au + bv = c \tag{4.7}$$

When $a \neq 0$ and $b \neq 0$,

$$uv = \frac{1}{a}(cv - bv^2) \tag{4.8}$$

The extreme values of this quadratic function are possibly to be reached at the two end points or when $v = c/2b$. When $a = 0$ or $b = 0$, the extreme values are only possible at the two end points. Further, since the line segments are end-to-end connected, for each segment, we only need to check one end point and one interval point that corresponds to $v = c/2b$.

Suppose the extremums are found to be

$$\min(uv) = p \quad , \quad \max(uv) = q,$$

then we obtain the bounding plane parameters C_1 and C_2 as

$$C_1 = q - x_0 y_0 \quad , \quad C_2 = p - x_0 y_0. \tag{4.9}$$

The resulting affine interval $\hat{z}$ for the multiplication is

$$\begin{aligned} \hat{z} &= A\hat{x} + B\hat{y} + \frac{C_1 + C_2}{2} + \frac{C_1 - C_2}{2}\varepsilon_k \\ &= y_0\hat{x} + x_0\hat{y} - x_0 y_0 + \frac{p+q}{2} + \frac{q-p}{2}\varepsilon_k, \end{aligned} \tag{4.10}$$

where ε_k is a new noise symbol indicating the uncertainty of the approximation error.

With a very similar procedure, we can trace the perimeter of U^{xy} and find the enforced bounds for z. The main difference is that U^{xy} is symmetric around (x_0, y_0), instead of $(0, 0)$. Now, each

side of the polygon can be written as

$$y = y_0 + \frac{c - a(x - x_0)}{b}, \tag{4.11}$$

When $a \neq 0$ and $b \neq 0$,

$$\begin{aligned} xy &= y_0 x + \frac{cx - ax(x - x_0)}{b} \\ &= -\frac{a}{b}x^2 + (y_0 + \frac{c + ax_0}{b})x. \end{aligned} \tag{4.12}$$

So, the extreme values of xy on each side must be found at two of the three points : the two end points and one internal point at

$$x = \frac{ax_0 + by_0 + c}{2a} \tag{4.13}$$

When $a = 0$ or $b = 0$, the extreme values must be at the two end points. Thus, by tracing along the perimeter of U^{xy} and checking the value of xy at one end point and one internal point on each side, we obtain the enforced bounds of z,

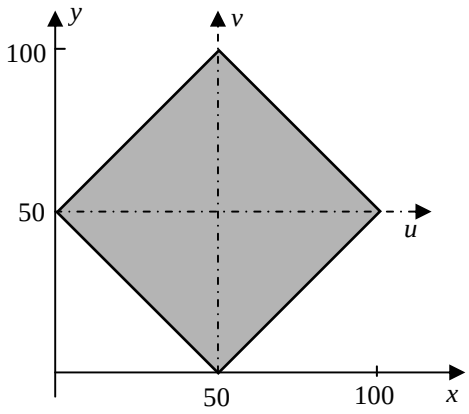
$$z_l = \min(xy), \text{ and } z_h = \max(xy).$$

The computational cost of the new multiplication algorithm is mainly in two phases. The first phase is to construct the polygon based on the input affine intervals. As we have presented Section 3.2.2, the complexity of this phase is $O(MN)$, where M is the number of shared noise symbols between $\hat{x}$ and $\hat{y}$, and N is the total number of noise symbols in the two inputs. The second phase is to trace the polygon to find the extreme values of uv and xy. Since the number of edges of a polygon is proportional to M, the complexity of polygon tracing is $O(M)$. So the total time complexity of the new multiplication algorithm is $O(MN+M) = O(MN)$. In the worst case, i.e., when $M = N$, the complexity is $O(N^2)$.

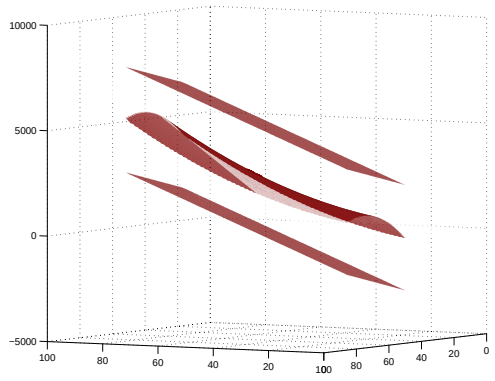
Lastly, we use an example to demonstrate the advantage of the new multiplication algorithm. The two inputs of the multiplication are

$$\hat{x}^a = \{50 + 25\varepsilon_1 + 25\varepsilon_2, [0, 100]\}$$
$$\hat{y}^a = \{50 + 25\varepsilon_1 - 25\varepsilon_2, [0, 100]\}.$$

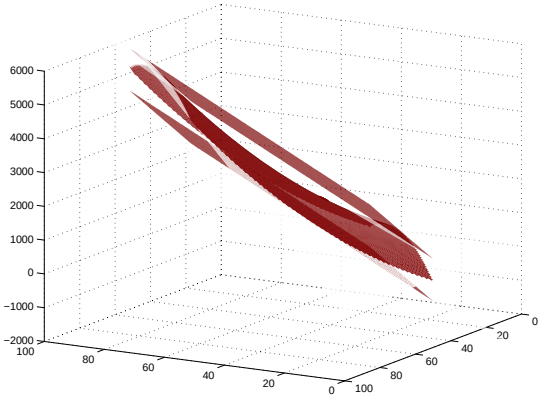
They share two noise symbols and do not have *effective* enforced bounds, i.e., the bounds $[0, 100]$ are derived from the center-symmetric affine intervals and do not add additional information about



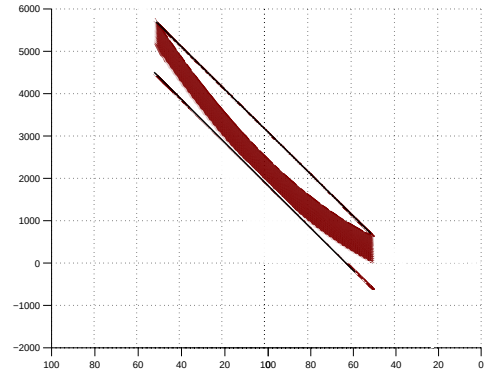
(a) Joint range of x and y



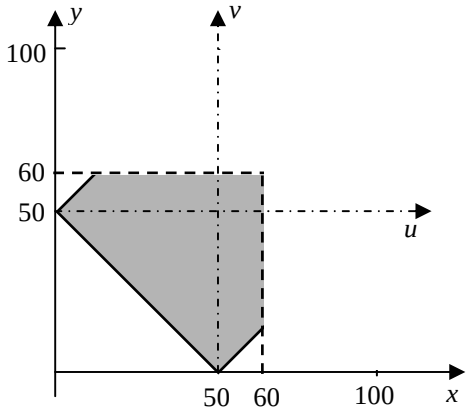
(b) Result by the original algorithm



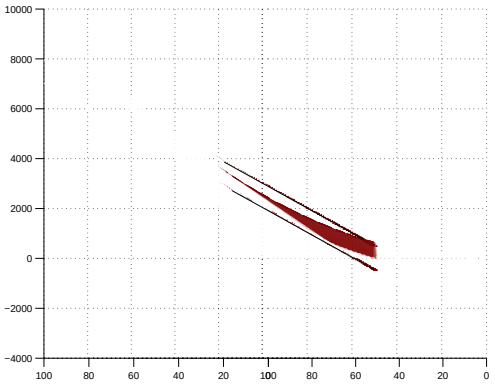
(c) New result



(d) New result (alternative view)



(e) Joint range of x and y with bounds



(f) New result for bounded inputs

In this example, $\hat{x} = 50 + 25\varepsilon_1 + 25\varepsilon_2$, $\hat{y} = 50 + 25\varepsilon_1 - 25\varepsilon_2$, and $z = xy$.
In (e), the enforced bounds are $x_h = 60$ and $y_h = 60$.

Figure 4.13: An example of the new multiplication algorithm

the intervals. The joint range U^{xy} is shown in Figure 4.13(a). We first show the result by the original multiplication algorithm. Using the *trivial range estimation*, we compute the resulting affine interval as

$$\begin{aligned}\hat{z} &= x_0\hat{y} + y_0\hat{x} - x_0y_0 + (\sum_i |x_i|)(\sum_i |y_i|)\varepsilon_k \\ &= 2500 + 2500\varepsilon_1 + 2500\varepsilon_k.\end{aligned}$$

From a geometric viewpoint, this result corresponds to the space between the following two parallel planes:

$$50x + 50y = 5000$$
$$50x + 50y = 0.$$

They are depicted in Figure 4.13(b). It is obvious that the two planes do not bound the nonlinear surface more tightly. There is still room for improvement. Next, we show the result by the new multiplication algorithm. By tracing uv on the four line segments, we obtain the two bounding planes:

$$50x + 50y = 3125$$
$$50x + 50y = 1875.$$

They are shown in Figure 4.13(c). An alternative view of these planes is in Figure 4.13(d). It is clear that the two bounding planes bound the nonlinear surface tightly, and the space between them is minimized. The resulting affine interval in this case is

$$\hat{z} = 2500 + 2500\varepsilon_1 + 625\varepsilon_k.$$

In addition, we also compute the enforced bounds for z by tracing xy on the four lines. They are

$$z_l = 0, \text{ and } z_h = 5625,$$

which is asymmetric to the center 2500. So the final answer is

$$\hat{z}^a = \{2500 + 2500\varepsilon_1 + 625\varepsilon_k, [0, 5625]\}.$$

Suppose instead that the two inputs have effective enforced bounds as the following,

$$[x] = [0, 60]$$
$$[y] = [0, 60],$$

their joint range now becomes an asymmetric polygon (see Figure 4.13(e)). Due to the new enforced bounds of the inputs, the output z has less variation, and hence the two bounding planes for the affine interval become even closer, as depicted in Figure 4.13(f). The new result in this case is

$$\hat{z}^a = \{2500 + 2500\varepsilon_1 + 425\varepsilon_k, [0, 3025]\}.$$

Case B—U^{xy} is a full or partial ellipse

If each input affine interval has a large number of noise symbols or the distributions of the noise symbols are normal, we can take advantage of the probabilistic interpretation of a 2D affine interval. In this case, the polygon is replaced by an ellipse that bounds the joint range with a high probability. When the inputs have enforced bounds, the joint range could be a partial ellipse, i.e., an ellipse that intercepts a range rectangle. Accordingly, the minivolume approximation is to find the two parallel planes that tightly bound the part of the nonlinear surface that *projects onto the ellipse*, not the entire polygon. This leads to a less pessimistic solution when many noise symbols are involved. Furthermore, constructing and tracing an ellipse is more computationally efficient than working on a polygon.

Let's first assume U^{xy} and U^{uv} are full ellipses. For the first step, i.e., the minivolume approximation, we trace the product of uv along the perimeter of the ellipse in the u–v plane, which has the form of

$$au^2 + bv^2 + cuv = 1, \tag{4.14}$$

where the parameters a, b and c can be derived from the two input affine intervals $\hat{x}$ and $\hat{y}$ (details have been presented in Section 3.2.2). We rewrite the ellipse equation as

$$u = -\frac{c}{2a}v \pm \frac{1}{2a}\sqrt{(c^2 - 4ab)v^2 + 4a}. \tag{4.15}$$

So uv can be expressed as a single variable function

$$uv = -\frac{c}{2a}v^2 \pm \frac{v}{2a}\sqrt{(c^2 - 4ab)v^2 + 4a} \tag{4.16}$$

$u_1 = \frac{1}{2a}(1 - c/\sqrt{4ab - c^2})\sqrt{2a - c\sqrt{a/b}}$ $v_1 = \sqrt{(2a - c\sqrt{a/b})/(4ab - c^2)}$
$u_2 = \frac{1}{2a}(-1 - c/\sqrt{4ab - c^2})\sqrt{2a - c\sqrt{a/b}}$ $v_2 = -\sqrt{(2a - c\sqrt{a/b})/(4ab - c^2)}$
$u_3 = \frac{1}{2a}(1 - c/\sqrt{4ab - c^2})\sqrt{2a + c\sqrt{a/b}}$ $v_3 = \sqrt{(2a + c\sqrt{a/b})/(4ab - c^2)}$
$u_4 = \frac{1}{2a}(-1 - c/\sqrt{4ab - c^2})\sqrt{2a + c\sqrt{a/b}}$ $v_4 = -\sqrt{(2a + c\sqrt{a/b})/(4ab - c^2)}$

Table 4.1: The four roots of equation (4.17)

Then we can find the extreme values of uv by solving

$$\frac{d(uv)}{dv} = 0. \tag{4.17}$$

This involves solving a 4th-degree equation. The derivation of the solution is lengthy and appears in Appendix B. We provide the four roots of equation (4.17) in Table 4.2.4. Finally, the extreme values of uv are determined by evaluating it at the four roots, illustrated in Figure 4.14(a).

If the inputs have enforced bounds, the perimeter of U^{xy} is composed of a partial ellipse and fewer than four horizontal or vertical line segments. When tracing the perimeter, we only need to evaluate the roots that are within the bounds, and those intersections between the ellipse and the bounds (see Figure 4.14 (b)).

Similarly, we find the enforced bounds of z by tracing xy along the perimeter of the full or partial ellipse that describes U^{xy}. A full ellipse in the x–y plane is written as

$$a(x - x_0)^2 + b(y - y_0)^2 + c(x - x_0)(y - y_0) = 1, \tag{4.18}$$

or

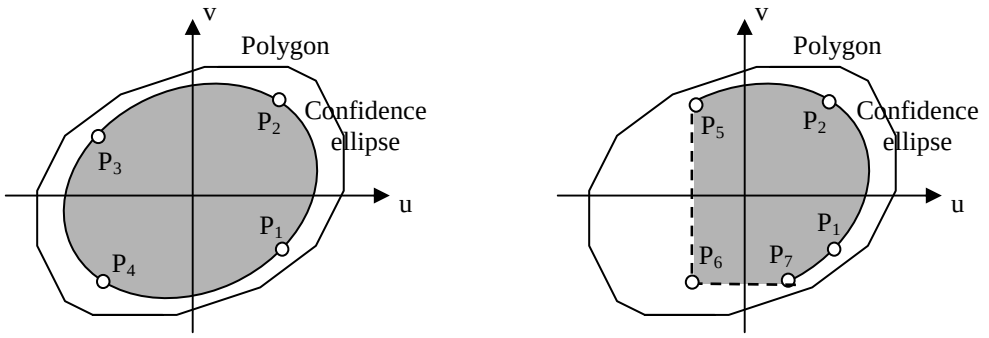
$$x = x_0 - \frac{c}{2a}(y - y_0) \pm \frac{1}{2a}\sqrt{(c^2 - 4ab)(y - y_0)^2 + 4a}. \tag{4.19}$$

Therefore

$$xy = x_0 y - \frac{c}{2a}y(y - y_0) \pm \frac{y}{2a}\sqrt{(c^2 - 4ab)(y - y_0)^2 + 4a}. \tag{4.20}$$

We can find the extremums of xy along the ellipse by letting

$$\frac{d(xy)}{dy} = 0. \tag{4.21}$$



(a) Tracing a full ellipse
(P_1—P_4 are the four roots)

(b) Tracing a partial ellipse
(P_5—P_7 are the intersections between the ellipse and the enforced bounds)

Figure 4.14: Ellipse tracing to find the extremes of uv

With the probabilistic interpretation, the joint range of u and v is reduced from a polygon to a confidence ellipse. (a) is for tracing a full ellipse, and (b) is for tracing a partial ellipse constrained by the enforced bounds. Evaluating uv at the points P_i's gives the extreme values needed for the minivolume approximation.

Again, this leads to a rooting-finding problem of a 4th-degree equation. The equation has no more than four real roots. Details on the 4th-degree equation are given in Appendix C. By evaluating xy at the roots that are within the bounds and the intersections between the ellipse and the bounding lines, we can determine the enforced bounds of z.

Tracing an ellipse has a clear advantage in computational complexity, compared to tracing a polygon, because the number of points that need to be evaluated is small, and is independent of the number of noise symbols. Hence, the main computational cost is in constructing the confidence ellipse. As we have presented in Section 3.2.2, this procedure involves computing the inputs' standard deviations and the correlation coefficient, which takes $O(N)$ time, with N being the total number of noise symbols. Therefore the total run time is $O(N + 1) = O(N)$, as opposed to $O(N^2)$ in the polygon case.

Multiplication algorithm summary

Finally, we summarize the complete multiplication algorithm in Figure 4.15. In the description, U^{xy} stands for the joint range of x and y, and U^{uv} stands for the joint range of u and v, where $u = x - x_0$

and $v = y - y_0$. The input affine interval $\hat{x}$ and $\hat{y}$ have N_1 and N_2 noise symbols, respectively.

4.2.5 Division on asymmetric affine intervals

Another important binary operation is division, i.e. the evaluation of $z = x/y$, given the asymmetric affine form $\hat{x}^a = \{\hat{x}, [x]\}$ and $\hat{y}^a = \{\hat{y}, [y]\}$ for the operands x and y. We assume the range of the denominator y does not include zero, in order for the division to be valid. The division algorithm published in [22] computes $\hat{z}$ indirectly by performing a multiplication of $\hat{x}$ with the reciprocal of $\hat{y}$, thus increasing the approximation error. Using the minivolume approximation, we develop a new division algorithm that *directly* computes the quotient, and at the same time, offers higher accuracy. The new algorithm follows the two-step procedure as we have outlined in Section 4.2.3: first to compute the affine interval $\hat{z}$ using the minivolume approximation, and then to find the enforced bounds $[z]$ by evaluating the extreme values of x/y over the joint range U^{xy}. In this section, we first introduce the mechanics of these two steps, and then dive into the mathematical details for each of the two cases based on the shape of the joint range U^{xy}.

The key of the minivolume approximation is to find the two parallel planes, $z_1 = Ax + by + C_1$ and $z_2 = Ax + By + C_2$, that tightly bound the nonlinear surface described by the target function. For division, the plane parameters A and B are

$$A = \frac{\partial(x/y)}{\partial x}(x_0, y_0) = \frac{1}{y_0} \quad , \quad B = \frac{\partial(x/y)}{\partial y}(x_0, y_0) = -\frac{x_0}{y_0^2},$$

and the distance function is

$$\frac{x}{y} - (Ax + By) = \frac{x}{y} - \frac{x}{y_0} + \frac{x_0}{y_0^2}y \tag{4.22}$$

which is a more complex function than the counterpart of multiplication. To find the remaining two parameters C_1 and C_2, we need to compute the minimal and the maximal values of the distance function in (4.22) over the joint range of x and y. We now prove that the extreme values of the distance function must be reached on the perimeter of U^{xy}. Suppose (x_1, y_1) is an internal point in U^{xy}. Then on the perimeter, there must exist two points (x_2, y_1) and (x_3, y_1) such that $x_2 \leq x_1 \leq x_3$. The distance function at (x_1, y_1) is written as

$$D(x_1, y_1) = (\frac{1}{y_1} - \frac{1}{y_0})x_1 + \frac{x_0}{y_0^2}y_1 \tag{4.23}$$

Input: $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i,\ [x_l, x_h]$

Input: $\hat{y} = y_0 + \sum_{i=1}^{N} y_i \varepsilon_i,\ [y_l, y_h]$

Output: $\hat{z} = z_0 + \sum_{i=1}^{N} z_i \varepsilon_i + \varepsilon_{N+1},\ [z_l, z_h]$

mult ($\hat{x}$, x_l, x_h, $\hat{y}$, y_l, y_h){

if $\min(N_1, N_2)$ is small

Construct U^{xy} as a polygon

$U^{uv} = U^{xy}$ shifted by $(-x_0, -y_0)$

$p = \min(uv)$ over U^{uv} by polygon tracing

$q = \max(uv)$ over U^{uv} by polygon tracing

$z_l = \min(xy)$ over U^{xy} by polygon tracing

$z_h = \max(xy)$ over U^{xy} by polygon tracing

else

Construct U^{xy} as a full or partial ellipse

$U^{uv} = U^{xy}$ shifted by $(-x_0, -y_0)$

$p = \min(uv)$ over U^{uv} by tracing a bounded ellipse

$q = \max(uv)$ over U^{uv} by tracing a bounded ellipse

$z_l = \min(xy)$ over U^{xy} by tracing a bounded ellipse

$z_h = \max(xy)$ over U^{xy} by tracing a bounded ellipse

$\hat{z} = y_0\hat{x} + x_0\hat{y} - x_0 y_0 + \frac{p+q}{2} + \frac{q-p}{2}\varepsilon_{N+1}$

compute the implied probabilistic bounds $\underline{z}_\lambda$ and $\overline{z}_\lambda$

$z_l = \max(z_l, \underline{z}_\lambda)$

$z_h = \min(z_h, \underline{z}_\lambda)$

output $\{\hat{z}, z_l, z_h\}$

}

Figure 4.15: The improved algorithm for the multiplication on asymmetric affine intervals

If $(1/y_1 - 1/y_0) \geq 0$,

$$D(x_2, y_1) \leq D(x_1, y_1) \leq D(x_3, y_1). \tag{4.24}$$

Otherwise,

$$D(x_2, y_1) \geq D(x_1, y_1) \geq D(x_3, y_1). \tag{4.25}$$

Hence, the extremums of the distance function $D(x, y)$ must be reached at a point on the perimeter. Therefore the main task in the minivolume approximation is to trace $D(x, y)$ along the perimeter of the joint range U^{xy}.

In the second step, we compute the enforced bounds of z to mitigate the overshoot problem. As we have mentioned, the ideal enforced bounds would be the "probabilistic" extreme values of x/y. However, its feasibility is dependent on whether we can efficiently compute the distribution of x/y. Luckily, when the joint range U^{xy} can be regarded as an ellipse, we find an analytical solution to the distribution, and hence the "probabilistic" extreme values of x/y. For the other situations, we still use the extreme values of x/y as the enforced bounds for z. Similar to the argument in the first step, we can easily prove that the extremums of x/y must be on the perimeter of the joint range. So when computing the "probabilistic" extreme values is not feasible, we find the enforced bounds by tracing x/y along the perimeter of U^{xy}.

Case A—U^{xy} is a polygon

When the joint range is a polygon, the first thing is to evaluate the distance function on each side of the polygon and find out the extremums. Any side of a polygon can be written as

$$ax + by + c = 0. \tag{4.26}$$

When $a = 0$ or $b = 0$, the extremums of the distance function are possible only at the two ends of the line segment. Otherwise, we rewrite the line equation as

$$x = \frac{1}{a}(c - by) \tag{4.27}$$

and hence the distance function on the line equals

$$\begin{aligned} D(x,y) &= \frac{x}{y} - \frac{x}{y_0} + \frac{x_0}{y_0^2}y \\ &= (\frac{1}{y} - \frac{1}{y_0})\frac{c-by}{a} + \frac{x_0}{y_0^2}y \\ &= my + \frac{n}{y} + l, \end{aligned} \tag{4.28}$$

where

$$\begin{aligned} m &= \frac{x_0}{y_0^2} + \frac{b}{ay_0} \\ n &= \frac{c}{a} \\ l &= -(\frac{b}{a} + \frac{c}{ay_0}) \end{aligned} \tag{4.29}$$

We can find the extremums by letting

$$\frac{dD(x,y)}{dy} = 0.$$

The answer is, if $m \cdot n \leq 0$, the extremums are reached at the two end points, and otherwise, the extremums are possible at two additional internal points where

$$y = \pm\sqrt{n/m}. \tag{4.30}$$

Therefore, on each side of the polygon, the distance function is evaluated at 2–4 points.

Suppose the extreme values of the distance function are found to be

$$C_1 = \max_{U^{xy}} D(x,y) \quad , \quad C_2 = \min_{U^{xy}} D(x,y).$$

The resulting affine interval for the quotient x/y is

$$\hat{z} = \frac{\hat{x}}{y_0} - \frac{x_0}{y_0^2}\hat{y} + \frac{C_1+C_2}{2} + \frac{C_1-C_2}{2}\varepsilon_k \tag{4.31}$$

Let's now consider the second step, i.e., to compute the extreme values of x/y by tracing the perimeter of U^{xy}. When a side of the polygon is parallel to the x-axis or the y-axis, the extremums of x/y are possible only at the two ends of the line segment. Otherwise, x/y on a side equals

$$\begin{aligned} x/y &= \frac{\frac{1}{a}(c-by)}{y} \\ &= \frac{1}{a}(\frac{c}{y} - b) \end{aligned} \tag{4.32}$$

whose extremums are possible only at the ends of the line segment as well. Therefore, by evaluating x/y on the vertices of the polygon, we obtain the enforced bounds of z,

$$z_l = \min(x/y), \text{ and } z_h = \max(x/y).$$

The computational cost of the new division algorithm is $O(MN)$, where M is the number of shared noise symbols between $\hat{x}$ and $\hat{y}$, and N is the total number of noise symbols in the two inputs, because constructing the polygon takes $O(MN)$ and tracing it for the extremums takes $O(M)$. In the worst case where the two inputs share every noise symbol, the complexity is $O(N^2)$.

Lastly, we apply the new division algorithm to an example and compare the result against the one using the original algorithm. The two inputs of the division are

$$\hat{x}^a = \{100 + 25\varepsilon_1 + 25\varepsilon_2, [50, 150]\}$$
$$\hat{y}^a = \{100 + 25\varepsilon_1 - 25\varepsilon_2, [50, 150]\}$$

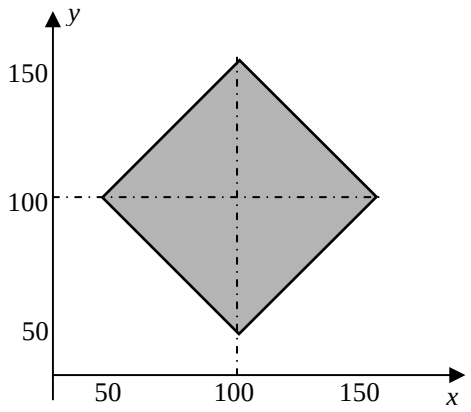
which share two noise symbols. Since there are no effective enforced bounds, their joint range is a symmetric polygon, as shown in Figure 4.16(a). The results from the original division algorithm [22] is in Figure 4.16(b). It is clear that this solution is very pessimistic, leaving a lot of reducible space between the bounding planes and the nonlinear surface. In contrast, the result from the new algorithm provides much tighter bounding planes, shown in Figure 4.16(c) and 4.16(d).[2]

Suppose instead that the two inputs in this example now have the following effective enforced bounds

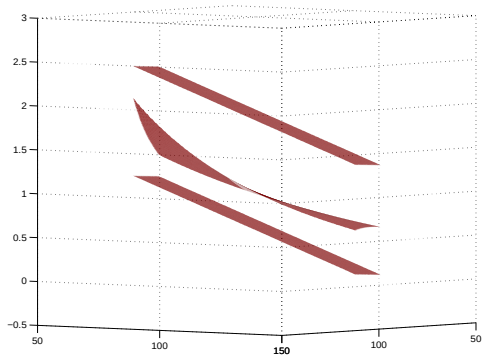
$$[x] = [50, 120]$$
$$[y] = [80, 150].$$

The joint range U^{xy} is reduced to an asymmetric polygon, as shown in Figure 4.16(e). The reduction in U^{xy} results in less uncertainty in the output z, and hence move the bounding planes of the affine interval $\hat{z}$ even closer (see Figure 4.16(f)).

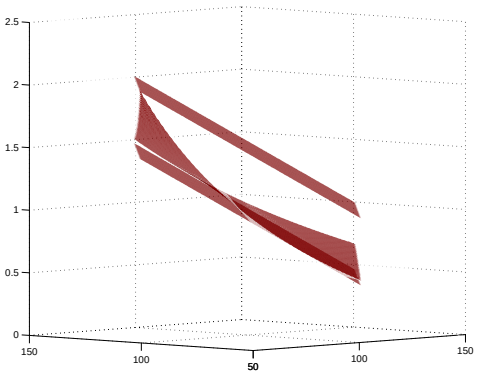
[2]One may notice in 4.16(d) that even with the new algorithm, there still remains space between the bounding planes and the nonlinear surface. It is obvious that if the two planes do not have to be parallel, we can find two planes that bound the nonlinear surface much tighter. However, it is the definition of an affine interval that requires the two bounding plane to be parallel: the space between the two planes is described by an affine interval, and the two bounding planes are reached when a certain noise symbols reach 1 and -1 and all other plane parameters are the same.



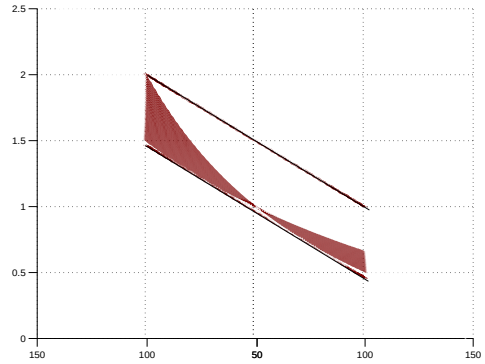
(a) Joint range of x and y



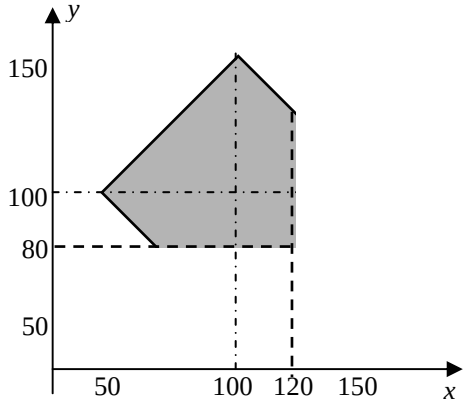
(b) Result from the original algorithm



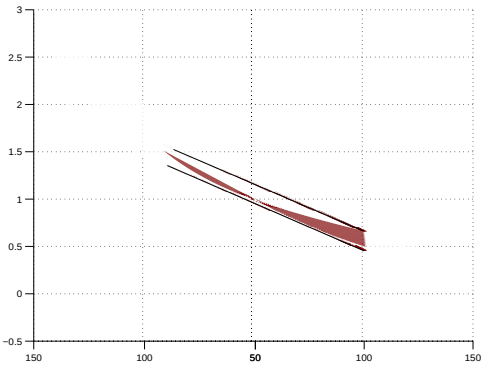
(c) New result



(d) New result (alternative view)



(e) Joint range of x and y with bounds



(e) New result for bounded inputs

Figure 4.16: Division by the minivolume approximation

In this example, $\hat{x} = 100 + 25\varepsilon_1 + 25\varepsilon_2$, $\hat{y} = 100 + 25\varepsilon_1 - 25\varepsilon_2$, and $z = x/y$. In (e), the enforced bounds are $x_h = 120$ and $y_l = 80$.

Case B—U^{xy} is a full or partial ellipse

When there is a larger number of noise symbols in each affine form of the inputs or the distributions of the noise symbols are normal, the joint range U^{xy} can be reduced from a polygon to a confidence ellipse. The enforced bounds of the inputs may further cut it down to a partial ellipse. Consequently, the new division algorithm need to be adjusted to reflect the change in the joint range.

For the sake of simplicity, we first assume U^{xy} is a full ellipse. For the minivolume approximation, we trace the value of the distance function $D(x, y)$ along the ellipse, which is described by

$$a(x - x_0)^2 + b(y - y_0)^2 + c(x - x_0)(y - y_0) = 1, \tag{4.33}$$

or

$$x = x_0 - \frac{c}{2a}(y - y_0) \pm \frac{1}{2a}\sqrt{(c^2 - 4ab)(y - y_0)^2 + 4a}. \tag{4.34}$$

Then the distance function on the ellipse can be written as a function of y

$$\begin{aligned} D(x, y) &= \frac{x}{y} - \frac{x}{y_0} + \frac{x_0}{y_0^2}y \\ &= (\frac{1}{y} - \frac{1}{y_0})(x_0 - \frac{c}{2a}(y - y_0) \pm \frac{1}{2a}\sqrt{(c^2 - 4ab)(y - y_0)^2 + 4a}) + \frac{x_0}{y_0^2}y. \end{aligned} \tag{4.35}$$

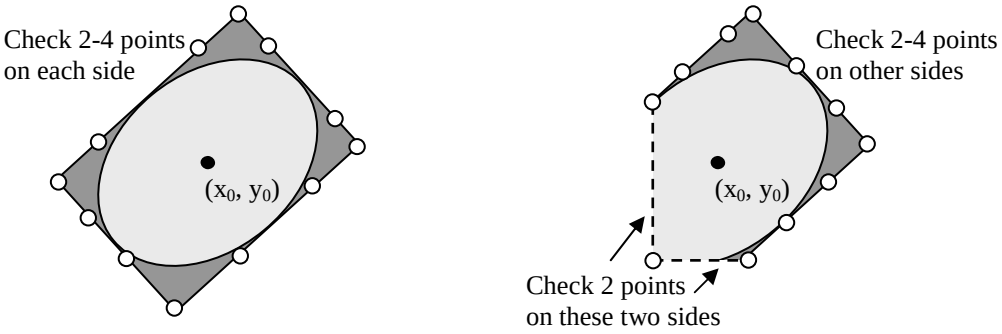
Theoretically, we can obtain the extreme values of $D(x, y)$ on the ellipse by solving

$$\frac{dD(x, y)}{dy} = 0.$$

Unfortunately, this leads to a 6th-degree equation solving problem, which does not have an analytical solution.

Therefore, we seek an approximate solution to the extremums of the distance function by conservatively simplifying the ellipse with a rectangular bounding box (see Figure 4.17(a)). The sides of the bounding box are parallel to the principle axes of the ellipse. Note that since a rectangle is a special polygon, the extremums of the distance function on the rectangle can be computed exactly the same way as in Case A. On each side of the rectangle, we need to evaluate the distance function at 2–4 points.

If there exist enforced bounds, we simply replace the true range U^{xy} with a conservative approximation, i.e., the polygon intersected by the rectangular bounding box and the enforced bounds



(a) Bounding box of a full ellipse (b) Bounding box of a partial ellipse

Figure 4.17: Tracing the bounding box of an ellipse

(see Figure 4.17(b)). Similarly, on each side of the polygon, the extremums of the distance function are possible at 2–4 points.

Next, we discuss how the enforced bounds of z are computed. Luckily, in this case, we are able to find the "probabilistic" extreme values of x/y through the knowledge on the distribution of x/y. As we know, when $\hat{x}$ and $\hat{y}$ have many noise symbols, their distributions can be regarded as normal distributions, and therefore the essential problem is to find out the distribution of the quotient of two normal random variables. By utilizing the results in [61], we obtain the CDF of z as the following,

$$P(z < s) = \begin{cases} \Phi(\frac{b(s-c_1)/c_2 - a}{\sqrt{1+((s-c_1)/c_2)^2}}) & \text{when } c_2 > 0 \\ 1 - \Phi(\frac{b(s-c_1)/c_2 - a}{\sqrt{1+((s-c_1)/c_2)^2}}) & \text{otherwise} \end{cases} \tag{4.36}$$

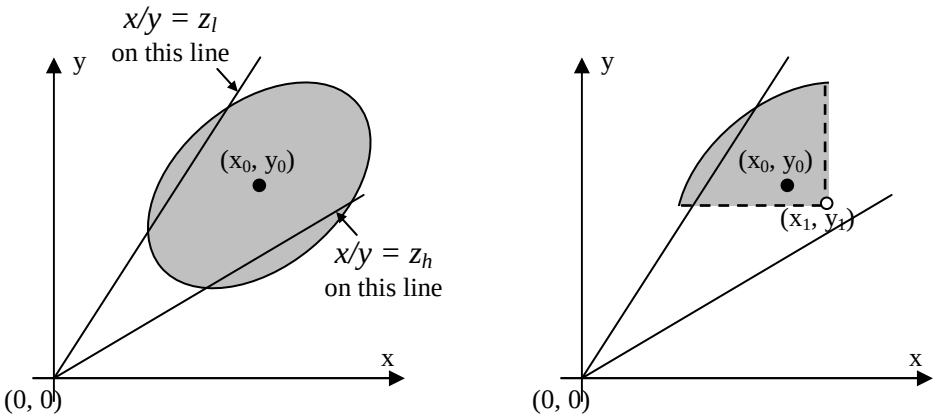
where $\Phi(x)$ is the standard normal CDF. The complete derivation and the constants c_1, c_2, and a are detailed in Appendix D.

From (4.36), we are able to find the probabilistic lower and upper bounds, z_l and z_h, that satisfy:

$$P(z < z_l) \leq \delta, \text{ and } P(z > z_h) \leq \delta,$$

where δ is a small number, for example, 0.05%. Note that on the x–y plane, the lower and upper bounds of x/y are reached on two straight lines, shown in Figure 4.18(a). They are $x/y = z_l$, and $x/y = z_h$.

However, when the inputs x and y have enforced bounds, it is possible that the z_l and z_h computed above may not be reached. An example is shown in Figure 4.18(b). In this case, the line



(a) U^{xy} is a full ellipse (b) U^{xy} is a partial ellipse

Figure 4.18: The enforced bounds for $z = x/y$

In (a), x/y reaches the probabilistic bounds z_l and z_h on the two straight lines. In (b), the joint range U^{xy} is a partial ellipse due to the enforced bounds of x and y. In this case, z_h is never reached, and therefore the upper bound should be adjusted to x_1/y_1.

$x/y = z_h$ is excluded from the joint range U^{xy}, and the true upper bounded is reached at (x_1, y_1) which is the intersection between two bounding lines. Therefore, when dealing with a partial ellipse, we should also compute the values of x/y at the intersections between any two bounding lines and adjust z_l and z_h accordingly.

Division algorithm summary

Finally, we summarize the complete division algorithm in Figure 4.19. In the description, U^{xy} stands for the joint range of x and y, and $D(x, y)$ is the distance function in (4.6). The input affine interval $\hat{x}$ and $\hat{y}$ have N_1 and N_2 noise symbols, respectively.

4.3 Experimental Results

In this section, we study the accuracy improvements afforded by the new techniques, mainly the asymmetric bounds, the minivolume approximation, and probabilistic bounding. We are especially interested in the applications that involve a large number of non-affine functions in a long computation chain, since this is the area of failure of the original affine interval techniques.

Input: $\hat{x} = x_0 + \sum_{i=1}^{N} x_i \varepsilon_i,\ [x_l, x_h]$

Input: $\hat{y} = y_0 + \sum_{i=1}^{N} y_i \varepsilon_i,\ [y_l, y_h]$

Output: $\hat{z} = z_0 + \sum_{i=1}^{N} z_i \varepsilon_i + \varepsilon_{N+1},\ [z_l, z_h]$

div ($\hat{x}$, x_l, x_h, $\hat{y}$, y_l, y_h){

if $\min(N_1, N_2)$ is small

construct U^{xy} as a constrained polygon

$C_1 = \max(D(x, y))$ over U^{xy} by polygon tracing

$C_2 = \min(D(x, y))$ over U^{xy} by polygon tracing

$z_l = \min(x/y)$ over U^{xy} by polygon tracing

$z_h = \max(x/y)$ over U^{xy} by polygon tracing

else

construct the bounding box of a confidence ellipse

construct U^{xy} as a constrained bounding box

$C_1 = \max(D(x, y))$ over U^{xy} by tracing the bounding box

$C_2 = \min(D(x, y))$ over U^{xy} by tracing the bounding box

compute z_l and z_h using the inverse CDF of z

$\hat{z} = \frac{\hat{x}}{y_0} - \frac{x_0}{y_0^2}\hat{y} + \frac{C_1+C_2}{2} + \frac{C_1-C_2}{2}\varepsilon_{N+1}$

compute the implied probabilistic bounds $\underline{z}_\lambda$ and $\overline{z}_\lambda$

$z_l = \max(z_l, \underline{z}_\lambda)$

$z_h = \min(z_h, \underline{z}_\lambda)$

output {$\hat{z}$, z_l, z_h}

}

Figure 4.19: The improved algorithm for the division on asymmetric affine intervals

To evaluate accuracy, we compare our interval results to simulation results which are obtained from 10^6 random samples. To understand how tight our interval results bound the simulation results, we define an accuracy measurement, called the *tightness ratio*, as the ratio between the estimated range and the simulated range. A large tightness ratio implies a loose bound. Normally speaking, a tightness ratio of 1.0 is most accurate. However, keep in mind that the intervals we estimate are probabilistic bounds. When an interval has a large number of uncertainty terms, its distribution is close to a normal distribution, and consequently, our estimated probabilistic bound is likely to be much narrower than the worst case range in the simulation. Therefore, when the tightness ratio is less than 1, the goodness of a bound should be measured on whether it captures the majority of the data points. We define our second accuracy measurement, called the *capture rate*, as the percentage of the simulated data that falls into the estimated bounds. 100% capture rate is the most desired accuracy. We use both of these two accuracy measurements in our experiments.

4.3.1 A multiplication chain

This experiment is on a chain of multiplications (described in (4.20)) on affine intervals, which is notorious for range explosion.

$$\textit{for i = 1:n} \quad x = x \cdot y \tag{4.37}$$

Since any approximation error that occurs at one stage is passed on to all subsequent stages, the deeper the computation depth is, the more pessimistic the interval becomes, if not handled carefully. Hence having an accurate interval multiplication algorithm is crucial to capture the interval growth in the multiplication chain.

In this section, we first experiment on a simple example where the inputs to the multiplication chain have only a small number (3 is used) of uniformly distributed noise terms. Note that in this case, probabilistic bounding does not take effect, because the conditions required by the Central Limit Theorem are not satisfied. Therefore, any accuracy improvement of the interval computations is a result of the use of asymmetric bounds and the minivolume criteria. Then, we complicate the example by increasing the number of noise terms in the inputs to 100. In this case, we expect to see additional accuracy improvement attributed to probabilistic bounding. In all the experiments, we compare our interval results to simulation results, obtained from 10^6 runs, and use tightness ratio

and/or capture rate as accuracy measurements.

In our first experiment, the inputs to the multiplication chain are initialized to be :

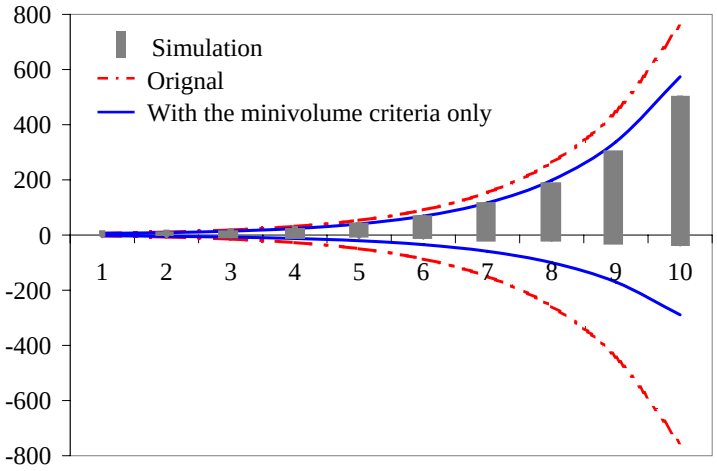
$$\begin{aligned} x &= 2 + \varepsilon_1 + 0.8\varepsilon_2 + 0.8\varepsilon_3 \\ y &= 1 - 0.2\varepsilon_1 + 0.1\varepsilon_2 + 0.4\varepsilon_4, \end{aligned} \tag{4.38}$$

where each ε_i is uniformly distributed in [-1 1]. To evaluate accuracy, we compare the growth of the range of x to the simulation results, obtained by randomly sampling ε_i's for 10^6 times and recording the minimal and the maximal values of x at each iteration stage. We show the improvements by the new algorithm in two steps: first we use only the minivolume criteria for the multiplication without enforcing asymmetric bounds, and then, we add asymmetric bounds and show the total improvements. Figure 4.20(a) shows the improvement by the minivolume criteria only. The x-axis indicates the number of iterations in the *for* loop, and the y-axis is for the value of x. The gray bars shows the interval growth captured during simulation, and the two dashed lines shows the estimated interval growth by the original AA algorithms from [22]. As we have expected, as more and more iterations are executed, the interval computation provides more and more pessimistic estimation. By using the minivolume criteria for multiplication, we are able to improve the results to a certain extent, shown by the two solid lines in the Figure 4.20(a). At the 10th iteration, the bound estimated by the original AA algorithms is 289% of the worst case range obtained by simulation, and with minivolume criteria, this ratio improves to 164%. The major problem is the lower bound of x is still too pessimistic, due to the overshoot problem. As we further apply the enforced asymmetric bounds, both the upper bound and the lower bound of x become more accurate (shown by the two solid lines in Figure 4.20(b)), and the improvement is more significant for those intervals with longer computation depth: at the 10th iteration, the tightness ratio (ratio between the estimated bound and the simulated range) is improved from the original 2.89 to 1.16.

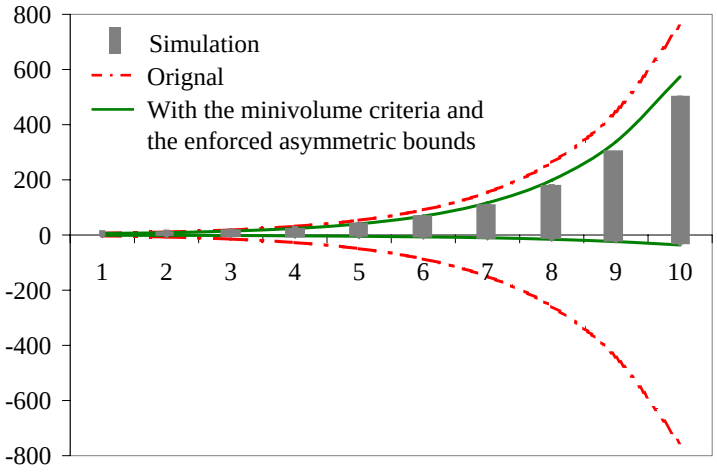
Next, we study whether correlation between the inputs have any effect on the accuracy. In our last experiment, the correlation coefficient between the inputs x and y equals -0.17. Now, we slightly change the inputs to (4.39), increasing the correlation coefficient to 0.93.

$$\begin{aligned} x &= 2 + \varepsilon_1 + 0.8\varepsilon_2 + 0.8\varepsilon_3 \\ y &= 1 + 0.2\varepsilon_1 + 0.4\varepsilon_2 + 0.4\varepsilon_3, \end{aligned} \tag{4.39}$$

We see similar accuracy improvements to those in the last experiment: at the 10th iteration, the



(a) With the minivolume criteria only



(b) With the minivolume criteria and the enforced bounds

Figure 4.20: Experimental results on a multiplication chain

The x-axis indicates the number of iterations in the *for* loop, and the y-axis is for the value of x. The minivolume approximation improves the interval estimation to a certain extent (in (a)), and by further enforcing asymmetric bounds, we more accurately capture the interval growth in the multiplication chain (in (b)).

No. of iterations	10	100	200	500	1000
Tightness ratio (Original)	2.89	5.93	8.25	14.09	N/A
Tightness ratio (New)	1.16	1.25	1.38	2.48	N/A

(a) 3 noise terms, weak correlation ($\rho = -0.17$)

No. of iterations	10	100	200	500	1000
Tightness ratio (Original)	2.14	4.52	8.17	21.17	N/A
Tightness ratio (New)	1.10	1.23	1.32	2.60	N/A

(b) 3 noise terms, strong correlation ($\rho = 0.93$)

Table 4.2: Accuracy comparison for more iterations

tightness ratio is improved from the original 2.14 to 1.10.

We are also interested in the accuracy comparison between the original AA and our new AA (without probabilistic bounding) as we run even more iterations. Table 4.2 summarizes the results for the cases with both weak and strong correlations for 10, 100, 200, 500, and 1000 iterations. For both cases, we have the following observations: 1) as a large number of multiplications are executed, the bounds estimated by the original AA become uselessly loose (at the 500th iteration, the bound is more than $14\times$ the simulated range); 2) Compared to the original AA, our new AA offers much slower growth of the bounds (at the 500th iteration, the bound is about $2.5\times$ the simulated range); 3) at the 1000th iteration, the intervals grow beyond the numerical range of double precision.

Now, we increase the number of noise terms in the inputs x and y to 100. The central values for x and y are 2.0 and 1.0, respectively, and the coefficient for each noise term are chosen randomly between [-0.05, 0.05]. In this case, since there are a large of noise terms in the affine intervals, probabilistic bounding takes place in all the interval computation algorithms. We notice that in the results, the estimated bounds are always narrower than the simulated range due to probabilistic bounding, or the tightness ratios are always smaller than 1.0. Therefore, we use capture rate as a better accuracy measure in this experiment. The accuracy results for 10, 20, 30, 40 and 50 iterations are shown in Table 4.3.1. We can see that up to 30 iterations, the capture rate is very high ($98.7\%+$), and beyond that, the capture rate drops significantly due to insufficient numerical precision. To explain that, we take the 50th iteration as an example. The inputs to the multiplication at this step

No. of iterations	10	20	30	40	50
Capture rate	99.999%	99.993%	98.706%	87.315%	72.515%

Table 4.3: Accuracy in the case of a large number noise terms

have extremely different ranges: the standard deviation of x is 8.9×10^{13}, whereas the standard deviation of y is only 0.458. Therefore, in the interval multiplication, the confidence ellipse that is constructed during the computation is almost a straight line, which causes a great loss of precision. Hence, when we conduct a binary operation (add, subtract, multiply, divide, etc.) on two intervals, we need to avoid such situations where one range is more than 10 orders of magnitude greater than the other.

In summary, the experiments on a multiplication chain demonstrate the advantages of our new AA over the original AA. The bounds are much tighter and grow at a much slower rate, as more and more computations are involved. Furthermore, the bounds are able to capture more than 98% of the data in most cases. We have also shown some extreme cases where our interval computations encounter numerical precision problems. These situations should be avoided in practical use of our interval method.

4.3.2 Cholesky Decomposition

Background

Our next experiment is on the *Cholesky decomposition*, one of the major matrix decomposition algorithms. Given a *symmetric positive definite* matrix A, there is a unique lower triangular matrix L with positive diagonal elements such that

$$A = L \cdot L^T \tag{4.40}$$

Symmetric means that $a_{ij} = a_{ji}$ for $i, j = 1, ..., N$, and positive definite means that

$$v' \cdot A \cdot v > 0 \text{ for all vectors v.} \tag{4.41}$$

An equivalent interpretation of positive definite is that A has all positive eigenvalues. Although symmetric positive definite matrices are rather special, they occur frequently in some applications, for

example, the covariance matrix of a set of random variables is usually symmetric positive definite. A primary use of the Cholesky decomposition is to solve positive definite linear systems [83].

This factorization is sometimes referred to as "taking the square root" of the matrix A. The "square root" L is computed as the following:

$$\begin{aligned} L_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} L_{ik}^2} \qquad i = 1, 2, ...N \\ L_{ji} &= \frac{1}{L_{ii}} \Big(a_{ij} - \sum_{k=1}^{i-1} L_{ik} L_{jk} \Big) \qquad j = i+1, i+2, ..., N \end{aligned} \tag{4.42}$$

We can see that the Cholesky decomposition heavily involves non-affine operations. In fact, for a matrix of size N, the Cholesky decomposition uses approximately $N^3/6$ multiplications and additions, $N^2/2$ divisions and N square root functions. Furthermore, these non-affine operations are cascaded in a long computation chain, since the computation of L_{ji} depends on the other elements in L that have already been computed. Therefore, performing accurate interval analysis on the Cholesky decomposition is very challenging and a nice test case for us.

Results

We perform the Cholesky decomposition on an interval matrix $\hat{A}$. To ensure that any sample of this interval matrix A is a positive definite matrix, we deliberately make central values of the diagonal elements much larger that those of the rest of the matrix. More specifically, we first construct the central matrix A_0 by assigning the central value of each element a_{0ij} to be:

$$a_{0ij} = \begin{cases} ki^2 & i = j \\ i & i < j \\ j & i > j \end{cases} \tag{4.43}$$

Note that $a_{0ij} = a_{0ji}$, and k is constant across the matrix and we set it to be 5 in most of the experiments in this section. An example of 5×5 central matrix with $k = 5$ is

$$A = \begin{bmatrix} 5 & 2 & 3 & 4 & 5 \\ 2 & 20 & 3 & 4 & 5 \\ 3 & 3 & 45 & 4 & 5 \\ 4 & 4 & 4 & 80 & 5 \\ 5 & 5 & 5 & 5 & 125 \end{bmatrix} \tag{4.44}$$

The determinant of this matrix equals 37729725 which is well above zero. We then add 30% variation to each element. Formally speaking, each element of the interval matrix A is

$$\hat{a}_{ij} = a_{0ij} + 0.3 a_{0ij} \varepsilon_{ij}.$$

To ensure that any sample of this interval matrix is a symmetric matrix, we constrain that ε_{ij} and ε_{ji} are fully correlated, or in other words, they are the same noise symbol. Each noise symbol is uniformly distributed in [-1, 1].

Cholesky decomposition of an interval matrix $\hat{A}$ results in an interval lower triangular matrix $\hat{L}$. To measure the accuracy of $\hat{L}$, we compare the upper and lower bounds of each element $\hat{L}_{ij}$ ($i > j$) to the simulation results, which are obtained by randomly sampling the interval matrix $\hat{A}$ 10^6 times and conducting Cholesky decomposition on every sample matrix. The two accuracy measurements, tightness ratio and capture rate, are evaluated for each element in $\hat{L}$.

Comparing to the original AA, we expect to see tighter bounds, as a result of two types of improvements discussed in this Chapter. One is algorithmic enhancements, including the use of asymmetric bounds and the minivolume approximation for nonlinear binary interval operations. The other type of improvements are attributed to probabilistic bounding, for both 1D and 2D affine intervals (we use confidence intervals for the 1D case and confidence ellipses for the 2D case). In this section, we first present the bounds estimated by adopting only the algorithmic enhancements, and show how much the tightness of the bounds are improved over those from the original AA. Then, we apply both types of improvements and show the accuracy of the new bounds.

In our first experiment, we conduct Cholesky decomposition on a 10×10 interval matrix as described above and study the improvements over the original AA. Figure 4.21(a) shows the tightness

ratio for each element in $\hat{L}$ when we use the original AA. The tightness ratios range from 1.0 to 1.78. In comparison, we show in Figure 4.21 (b) the tightness ratios when we adopt the algorithmic enhancements. They vary from 1.0 to 1.29. In other words, the worst case bound is improved from 1.78X real range to 1.29X real range.

Next, we apply both the algorithmic enhancements and probabilistic bounding, and measure the accuracy of the resulting interval matrix $\hat{L}$, characterized by the tightness ratios and capture rates for all the elements in $\hat{L}$. We conduct three sets of experiments with different values for two matrix parameters, namely, k and variation. In Experiment A, $k = 5$, which guarantees a well conditioned interval matrix (meaning the determinant of any sample matrix is always well above zero), and the variation of each element is 30%. The results are shown in Figure 4.22. The tightness ratios range from 0.96 to 1.04 (see Figure 4.22(a)). Due to the use of probabilistic bounding, the estimated bound is possible to be narrower than the real range, or, the tightness ratio is possible to be less than 1.0, as we can see in the results. When the tightness ratio is less than 1.0, a better accuracy measurement is the capture rate, i.e., the percentage of the simulated results that fall into the estimated bound. The capture rates for all the elements in $\hat{L}$ are shown in Figure 4.22(b), ranging from 99.99% to 100%, which means our estimated bounds capture at least 99.99% of the simulation results. In Experiment B, we reduce the k value from 5 to 1.5, [3] making the interval matrix less well conditioned (the determinant of A_0 is consequently reduced from 37729725 to 250), and keep the variation at 30%. The results are shown in Figure 4.23. The tightness ratios, ranging from 0.81 to 1.12, indicate that for some elements, the bounds are slightly looser than the those in Experiment A, and for others, the bounds are slightly tighter. The capture rates still remains very close to 100% (from 99.97% to 100%). In Experiment C, we restore the k value to 5, and increase the variations from 30% to 70%.[4] The results are shown in Figure 4.24. The tightness ratios range from 0.81 to 1.05, and the capture rates are above 99.975%. From these three experiments (the results are summarized in Table 4.4)[5], we can see that the accuracy of the interval Cholesky decomposition is very high across interval

[3] When we reduce k to 1.0, we find the interval matrix is ill conditioned such that some sample matrices are not positive definite.

[4] When we increase variation to 80%, we find the interval matrix is ill conditioned such that some sample matrices are not positive definite.

[5] We have also tried the combination of $k = 1.5$ and 70% variation, and find the interval matrix becomes ill conditioned such that some sample matrices are not positive definite

matrices with different conditions and variations.

	k	Variation	Tightness ratio (range)	Capture rate (range)
Experiment A	5	30%	$[0.96, 1.04]$	$[99.99\%, 100\%]$
Experiment B	1.5	30%	$[0.81, 1.12]$	$[99.97\%, 100\%]$
Experiment C	5	70%	$[0.81, 1.05]$	$[99.975\%, 100\%]$

Table 4.4: Summary of Experiment A, B, and C

Now, we experiment on a sparse matrix of size 1000×1000, and vary the number of non-zero elements in a row from 10 to 100. The diagonal elements are always set non-zero, and other non-zero elements are chosen randomly. For each non-zero element, the central value is assigned as described in (4.43), and the variation equals 30% of its central value. One nice thing about conducting decomposition on a sparse matrices is that due to a large number of zero elements, a lot of interval computations are conducted between a normal affine interval and a zero affine interval, which do not introduce any approximation, and in some cases (multiplication with a zero interval), even remove all the approximation errors made previously. This is evidenced by the high accuracy measurements shown in Table 4.5. We can see that as the number of non-zero elements increases (from 10 to 100), the capture rate only reduced slightly (from $99.89\%+$ to 98.45%+), although the tightness ratio (low end) drops significantly. Note that when tightness ratio is smaller than 1, capture rate is a more sensible accuracy measure, because as more and more noise symbols are involved, the probabilistic bounds with the same capture rate may become less and less tight compared to the real range.

Finally, we note that one needs to be careful about the condition of the matrix$\hat{A}$ when conducting interval Cholesky decomposition. In all the experiments we have shown so far, we deliberately choose well conditioned matrices. We do find that in certain less well conditioned cases, interval

No. non-zero elements in a row	Tightness ratio (range)	Capture rate (range)
10	$[0.78,\ 1.08]$	$[99.89\%,\ 100\%]$
50	$[0.63,\ 1.12]$	$[99.26\%,\ 100\%]$
100	$[0.52,\ 1.14]$	$[98.45\%,\ 100\%]$

Table 4.5: Accuracy of Cholesky decomposition on sparse matrices of size 1000×1000

$$\begin{bmatrix}
1 & & & & & & & & & \\
1.14 & 1.01 & & & & & & & & \\
1.14 & 1.31 & 1.02 & & & & & & & \\
1.14 & 1.31 & 1.42 & 1.02 & & & & & & \\
1.14 & 1.31 & 1.45 & 1.56 & 1.02 & & & & & \\
1.14 & 1.32 & 1.44 & 1.55 & 1.61 & 1.02 & & & & \\
1.14 & 1.3 & 1.44 & 1.55 & 1.61 & 1.67 & 1.02 & & & \\
1.14 & 1.31 & 1.43 & 1.55 & 1.6 & 1.67 & 1.71 & 1.02 & & \\
1.14 & 1.3 & 1.46 & 1.55 & 1.62 & 1.63 & 1.73 & 1.78 & 1.02 & \\
1.14 & 1.3 & 1.43 & 1.56 & 1.6 & 1.69 & 1.72 & 1.74 & 1.78 & 1.02
\end{bmatrix}$$

(a) Tightness ratios for all the elements in $\hat{L}$ for the original AA

$$\begin{bmatrix}
1 & & & & & & & & & \\
1 & 1 & & & & & & & & \\
1 & 1.03 & 1.01 & & & & & & & \\
1 & 1.05 & 1.11 & 1.01 & & & & & & \\
1 & 1.05 & 1.11 & 1.18 & 1.01 & & & & & \\
1 & 1.04 & 1.14 & 1.17 & 1.19 & 1.01 & & & & \\
1 & 1.05 & 1.12 & 1.19 & 1.21 & 1.22 & 1.01 & & & \\
1 & 1.06 & 1.12 & 1.18 & 1.22 & 1.24 & 1.26 & 1.01 & & \\
1 & 1.06 & 1.13 & 1.18 & 1.21 & 1.25 & 1.24 & 1.25 & 1.01 & \\
1 & 1.05 & 1.14 & 1.17 & 1.2 & 1.23 & 1.27 & 1.28 & 1.29 & 1.01
\end{bmatrix}$$

(b) Tightness ratios for all the elements in $\hat{L}$ for the new AA with only the algorithmic enhancements

Figure 4.21: Comparison of accuracy (tightness ratio) with the original AA

The worst case bound is improved from 1.78X real range to 1.29X real range

$$\begin{bmatrix}
1 & & & & & & & & & \\
1 & 1 & & & & & & & & \\
1 & 0.97 & 1.01 & & & & & & & \\
1 & 0.98 & 1 & 1.01 & & & & & & \\
1 & 0.97 & 0.98 & 1 & 1.01 & & & & & \\
1 & 0.96 & 0.97 & 1.01 & 1 & 1.01 & & & & \\
1 & 0.97 & 0.96 & 0.99 & 1.01 & 1.03 & 1.01 & & & \\
1 & 0.97 & 0.97 & 0.99 & 1.02 & 1.02 & 1.02 & 1.01 & & \\
1 & 0.97 & 0.99 & 1 & 1.02 & 1.03 & 1.02 & 1.03 & 1.01 & \\
1 & 0.97 & 0.98 & 1 & 0.99 & 1.03 & 1.04 & 1.03 & 1.02 & 1.01
\end{bmatrix}$$

(a) Tightness ratios for all the elements in $\hat{L}$

$$\begin{bmatrix}
100\% & & & & & & & & & \\
100\% & & & & & & & & & \\
100\% & 100\% & & & & & & & & \\
100\% & 99.991\% & 100\% & & & & & & & \\
100\% & 99.990\% & 99.995\% & 100\% & & & & & & \\
100\% & 99.992\% & 99.994\% & 99.996\% & 100\% & & & & & \\
100\% & 99.990\% & 99.993\% & 99.996\% & 99.998\% & 100\% & & & & \\
100\% & 99.991\% & 99.994\% & 99.997\% & 99.998\% & 99.999\% & 100\% & & & \\
100\% & 99.992\% & 99.993\% & 99.997\% & 99.998\% & 99.998\% & 100\% & 100\% & & \\
100\% & 99.990\% & 99.993\% & 99.996\% & 99.998\% & 99.998\% & 99.999\% & 99.999\% & 100\% & \\
100\% & 99.991\% & 99.994\% & 99.997\% & 99.998\% & 99.998\% & 99.999\% & 99.999\% & 100\% & 100\%
\end{bmatrix}$$

(b) Capture rates for all the elements in $\hat{L}$

Figure 4.22: Accuracy measurements for Experiment A

In experiment A, the matrix size is 10×10, $k = 5$, and each element in A has 30% variation. The new AA algorithms adopt both the algorithmic enhancements and probabilistic bounding

$$\begin{bmatrix}
1 & & & & & & & & & \\
1 & 1.01 & & & & & & & & \\
1 & 0.83 & 1.12 & & & & & & & \\
1 & 0.81 & 0.87 & 1.13 & & & & & & \\
1 & 0.8 & 0.88 & 0.92 & 1.13 & & & & & \\
1 & 0.81 & 0.89 & 0.93 & 0.95 & 1.12 & & & & \\
1 & 0.82 & 0.88 & 0.91 & 0.93 & 0.95 & 1.1 & & & \\
1 & 0.8 & 0.86 & 0.92 & 0.96 & 0.96 & 0.95 & 1.1 & & \\
1 & 0.82 & 0.87 & 0.92 & 0.93 & 0.95 & 0.94 & 0.94 & 1.09 & \\
1 & 0.82 & 0.88 & 0.92 & 0.93 & 0.96 & 0.97 & 0.95 & 0.98 & 1.08
\end{bmatrix}$$

(a) Tightness ratios for all the elements in $\hat{L}$

$$\begin{bmatrix}
100\% & & & & & & & & & \\
100\% & 100\% & & & & & & & & \\
100\% & 99.824\% & 100\% & & & & & & & \\
100\% & 99.826\% & 99.970\% & 100\% & & & & & & \\
100\% & 99.815\% & 99.972\% & 99.980\% & 100\% & & & & & \\
100\% & 99.814\% & 99.972\% & 99.980\% & 99.984\% & 100\% & & & & \\
100\% & 99.819\% & 99.972\% & 99.980\% & 99.981\% & 99.986\% & 100\% & & & \\
100\% & 99.822\% & 99.972\% & 99.980\% & 99.983\% & 99.986\% & 99.986\% & 100\% & & \\
100\% & 99.828\% & 99.970\% & 99.979\% & 99.986\% & 99.984\% & 99.988\% & 99.988\% & 100\% & \\
100\% & 99.823\% & 99.971\% & 99.982\% & 99.984\% & 99.985\% & 99.988\% & 99.991\% & 99.991\% & 100\%
\end{bmatrix}$$

(b) Capture rates for all the elements in $\hat{L}$

Figure 4.23: Accuracy measurements for Experiment B

In Experiment B, the matrix size is 10×10, $k = 1.5$, and each element in A has 30% variation.

$$\begin{bmatrix}
1.00 & & & & & & & & & \\
1.00 & 1.01 & & & & & & & & \\
1.00 & 0.88 & 1.05 & & & & & & & \\
1.00 & 0.81 & 0.98 & 1.05 & & & & & & \\
1.00 & 0.90 & 0.95 & 1.00 & 1.05 & & & & & \\
1.00 & 0.83 & 0.97 & 1.00 & 1.03 & 1.04 & & & & \\
1.00 & 0.86 & 0.97 & 1.03 & 1.02 & 1.03 & 1.04 & & & \\
1.00 & 0.87 & 0.99 & 0.99 & 1.00 & 1.03 & 1.02 & 1.03 & & \\
1.00 & 0.88 & 0.89 & 0.98 & 1.00 & 1.02 & 1.03 & 1.04 & 1.03 & \\
1.00 & 0.89 & 0.97 & 0.99 & 1.02 & 1.01 & 1.01 & 1.03 & 1.02 & 1.03
\end{bmatrix}$$

(a) Tightness ratios for all the elements in $\hat{L}$

$$\begin{bmatrix}
100\% & & & & & & & & & \\
100\% & 100\% & & & & & & & & \\
100\% & 99.991\% & 100\% & & & & & & & \\
100\% & 99.990\% & 100\% & 100\% & & & & & & \\
100\% & 99.990\% & 99.999\% & 99.997\% & 100\% & & & & & \\
100\% & 99.988\% & 99.999\% & 99.997\% & 99.994\% & 100\% & & & & \\
100\% & 99.990\% & 99.999\% & 99.997\% & 99.993\% & 99.990\% & 100\% & & & \\
100\% & 99.988\% & 99.999\% & 99.998\% & 99.992\% & 99.989\% & 99.983\% & 100\% & & \\
100\% & 99.990\% & 100\% & 99.997\% & 99.993\% & 99.989\% & 99.983\% & 99.982\% & 100\% & \\
100\% & 99.990\% & 99.999\% & 99.999\% & 99.992\% & 99.988\% & 99.985\% & 99.979\% & 99.975\% & 100\%
\end{bmatrix}$$

(b) Capture rates for all the elements in $\hat{L}$

Figure 4.24: Accuracy measurements for Experiment C

In Experiment C, the matrix size is 10×10, $k = 5$, and each element in A has 70% variation.

decomposition failed, even though all of the one million sample matrices are positive-definite. The following is an example:

$$A_0 = \begin{bmatrix} 12.8 & 4.6 & 4.6 \\ 4.6 & 7.1 & 3.8 \\ 4.6 & 3.8 & 9.3 \end{bmatrix}$$

and each element in $\hat{A}$ has 30% variation. The step that fails is an interval square root function on $[-1.8, 11.4]$. The real range of the input of this function is $[0.63, 11.08]$, obtained from simulation, with the low end close to zero. Therefore, it is very likely that the interval computations offer a slightly conservative estimate for the low end, which makes the subsequent square root function invalid.

This last example illustrates an important point about all interval-valued analysis methods. Just as the existence of high-precision 64-bit floating point arithmetic does *not* remove the need for delicate numerical analysis methods for numerically challenging problems, the existence of improved interval techniques does *not* mean that one can simply replace each scalar with an interval-valued counterpart, and expect perfect results. Estimation errors in intervals are analogous to finite precision errors in floating point, but much more macroscopic. Our asymmetric interval formulations and probabilistic bounding techniques greatly improve the "usability" of the affine model, but they are obviously not perfect. One obvious question is to see how much additional improvement can be squeezed from our model, at what cost per operator. But another more novel question is whether we might reformulate the numerics at the algorithm level specifically to mitigate some of these interval misestimation effects, just as is commonly done to deal with finite precision issues in ordinarily scalar numerical analysis. This seems an excellent avenue for future work in this area, but is beyond the scope of this thesis.

4.4 Summary

This chapter addresses a fundamental limitation of affine arithmetic, i.e., the interval represented by an affine form must be center-symmetric. This restriction highly limits the accuracy of nonlinear interval functions. We propose the notion of an *asymmetric affine interval*, which is an affine interval with enforced asymmetric bounds. These bounds also incorporate probabilistic information:

the probability of exceeding the asymmetric bounds is less than a user-specified small value. Then, based on this new representation, we develop interval computation algorithms for the six common nonlinear functions. The new algorithms propagate not only the affine intervals, but also the asymmetric bounds. In addition, for binary nonlinear functions, we introduce a better approximation method, called the *minivolume approximation*, and describe in detail the corresponding algorithms for multiplication and division. These efforts substantially improve the accuracy for affine arithmetic.

We also note the techniques described in this chapter, while allowing us to compute with asymmetric affine intervals, do not yet allow us to retrieve any sort of statistical description of the associated PDFs. In other words, up to now, we have used ideas from probability only to help us create tighter, approximated bounds on these intervals. In many applications, this is sufficient; anywhere one might use and be satisfied with an interval-valued model of an uncertainty, our asymmetric intervals may also be applied. We show two such applications in this chapter: one is a multiplication chain, and the other is the Cholesky decomposition. We will address the problem of retrieving some approximation of the underlying statistical distribution in the following chapter.

Chapter 5

Analyzing the Probability Distribution within an Asymmetric Affine Interval

The techniques we have developed so far in this thesis have improved the utility and accuracy of the affine interval representation, allowing us to better model range uncertainty in chains of linear and nonlinear computations. Hence, the next question is: can we "retrieve" from this interval representation itself, some useful approximation of the actual underlying PDF. The answer is yes. We develop the necessary approximations in this chapter, and show some practical applications of the ideas.

5.1 Motivation

Traditionally, interval techniques have been used to analyze the boundaries of the solution space of a problem. For instance, in our floating-point error analysis in Section 3.3, we use affine arithmetic to estimate the maximum floating-point error during a program execution. However, in many applications, knowing only the boundaries is not enough; detailed probability distribution within the boundaries is more desirable. Moreover, distributions in real applications are not always as regular as normal or uniform distributions, and sometimes are highly skewed. Hence, it calls for special techniques to analyze probability distributions.

There are several common approaches to distribution analysis. The first category is the analytical distribution estimation approach. Statistical Static Timing Analysis (SSTA) is a successful industry example of this approach [13, 23, 50, 54, 88]. The idea is to propagate normal distributions through each atomic operation and obtain a normal distribution for the target result. In SSTA, this means pushing correlated normal distributions representing signal arrival times through addition, subtraction, maximum, and minimum operators. Luckily, this works well in practice in the SSTA application. Linear operators (add, subtract) preserve normality. In the empirically important case of normal-valued circuit delays with variations that are not too dissimilar, there are classical normal approximations to the minimum and maximum of a pair of correlated normals [15] that work well. Unfortunately, the nonlinearities of max/min are not, in the general case, well modeled by normals: distributions with very different variances yield heavy-tailed PDFs in such cases. This is an unfortunately common problem, for example, representing a product or a quotient of correlated normals is not straightforward [61, 63]. Therefore, the effectiveness of this approach highly depends on the arithmetic operations involved in the application, and hence many techniques are application-specific.

Another example of a successful technique that relies on an explicit analytical form for the PDF is the recently proposed method of Asymptotic Probability Extraction (APEX, [53]). The technique relies on a novel connection between the time domain moments of any linear time invariant (LTI) circuit, and the moments of its distribution. A very efficient fitting technique, based on moment matching, can transform these time-domain moments into an efficient non-normal statistical model. The technique seems both general and efficient—at least, for any design object which has a "circuit-like" form, that is, an ordinary differential equation (ODE) form which can be simulated via time step integration, and from which LTI moments can be extracted. This is a usefully wide category of applications, yet, does not cover many of the DSP applications of interest to us in our own work. Many more general applications exist not as circuits, but as computational "recipes" in the form of C++ or MATLAB code. We need other techniques in such applications, which has motivated our original interest in interval-valued computation.

The second category is Monte Carlo simulation. By randomly sampling the solution space for a sufficiently large number of times, we can directly estimate the distribution [42, 94]. Monte Carlo methods are obviosly very general, and can be arbitrarily accurate, but very expensive with a large

application or a large number of random parameters. This motivates techniques ranging from basic confidence interval methods [6] to more sophisticated Design of Experiments (DOE) methodologies which seek to make optimal choices about how and where to sample these complex applications so as to make the most efficient statistical use of the resulting samples and to minimize the number of total samples required for any desired level of accuracy.

The third category includes those that sit in the middle of the spectrum between fully analytical approaches and Monte Carlo simulation. Histogram propagation is one of them [54]. It replaces distributions with histograms so that each uncertain quantity can be sampled from discrete bins. Instead of simulating on random samples, it simulates on the centers of the bins. Therefore, it is faster than Monte Carlo simulation, making exhaustive simulation feasible. The downside of this approach is that it is not capable of handling all types of correlations. Further, the time complexity increases rapidly with the number of inputs of an application. Another approach in this category is based on classical *matrix perturbation theory* and has been successfully applied to statistical interconnect modeling [56].

In this chapter, we explore how we can use the additional information carried in our asymmetric affine interval formulation to "retrieve" some empirically useful model of the PDF that underlies the range uncertainty modeled by the interval. We have shown that an affine interval with asymmetric bounding carries much more information than just the upper and lower bounds. It reveals the underlying components that affect the uncertainty of a quantity, and is especially powerful in handling correlations. So, can we enhance our interval techniques to analyze the probability distribution within an affine interval?

Let us first consider a simple problem where the application contains only a single arithmetic operator, $z = f(x)$, or $z = f(x, y)$, where the inputs x and y are in affine forms $x_0 + \sum_{i=1}^{N} x_i\varepsilon_i$ and $y_0 + \sum_{i=1}^{N} y_i\varepsilon_i$. In this example, let us assume ε_i's have the standard normal distributions $\mathcal{N}(0, 1)$, and hence, the input x and y are also normally distributed. Our goal in this problem is to estimate the probability distribution of z using affine interval techniques. We perform the corresponding interval computation on $\hat{x}$ and $\hat{y}$, and obtain the output in an asymmetric affine form $\hat{z}^a = \{z_0 + \sum_{i=1}^{N} z_i\varepsilon_i \ , \ z_l \ , \ z_h\}$. Let us ignore the asymmetric bounds z_l and z_h for now, and

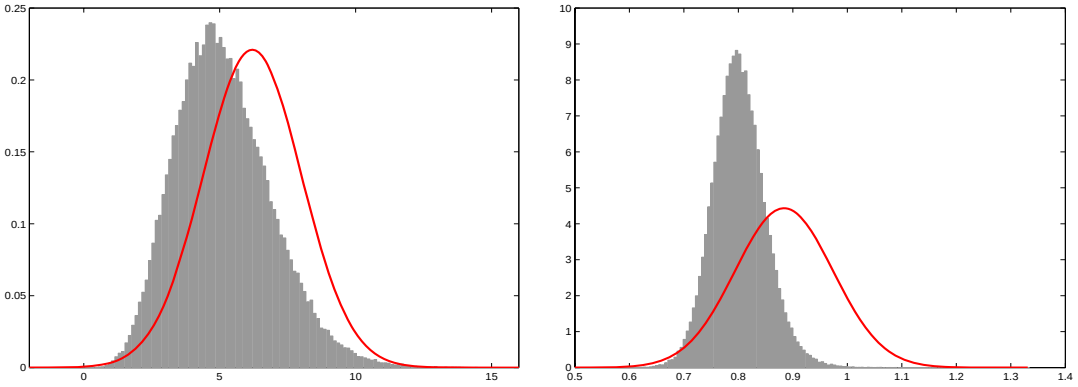
compute the mean and the standard deviation of the output through the affine form $\hat{z} = z_0 + \sum_{i=1}^{N} z_i \varepsilon_i$,

$$\mu_z = z_0$$
$$\sigma_z^2 = \sum_{i=1}^{N} z_i^2.$$

Now, the best we can do without using the asymmetric bounds is to fit a normal distribution curve based on the the first two moments. This approach works well for affine functions, namely, $x \pm y$, cx, and $x \pm c$, because the output of these affine functions on normal random variables is indeed a normal random variable. However, for non-affine functions, this approach may be highly inaccurate. We show the results for some non-affine functions, including $z = x \cdot y$, $z = x/y$, $z = \exp(x)$, $z = \log(x)$, $z = \sqrt{x}$, and $z = \frac{1}{x}$ in Figure 5.1. The algorithms for the interval computations are what we have presented in Chapter 4, except that we ignore the asymmetric bounds at the end when generating the PDF curves. In the figure, the histogram is obtained through Monte Carlo simulation with 10^5 samples (values are normalized such that the total area equals 1), and the curve is the probability density function generated using the mean and the standard deviation from the output affine interval. We can see that the PDF generated from the affine interval does not match the simulation result very well. Therefore, for applications that significantly involve non-affine functions, our current interval techniques are not sufficient to propagate distributions.

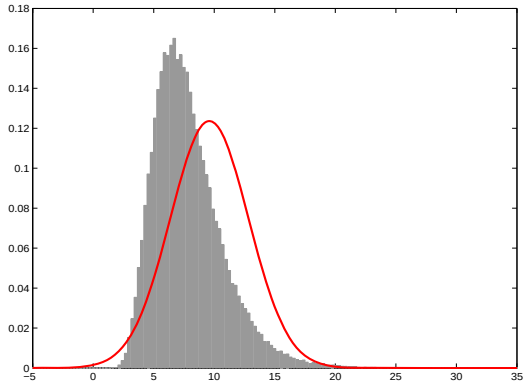
The results shown in Figure 5.1 reveal some common problems with the current interval computations when applied to distribution analysis. First, the central value of the resulting affine interval (i.e., the highest point in the PDF) does not coincide with the mode of the actual distribution (i.e., the point with the highest frequency in the histogram). The difference is especially significant for the $z = x/y$ function. Second, a normal distribution curve fails to capture the skewness, or the asymmetry, of the actual distribution. For non-affine functions, if the inputs have normal distributions, the output may not be normal any more, which is especially noticeable for the $z = x/y$, $z = \exp(x)$, $z = \log(x)$, and $z = \frac{1}{x}$ functions. Third, in the functions $z = x/y$ and $z = \frac{1}{x}$, the PDF's produced from the affine intervals have larger variations than the actual distributions. These problems are all related to the fact that a standard affine interval, without the enforced asymmetric bounds, is intrinsically symmetric, and so is the distribution produced directly from the affine form.

In the rest of this chapter, we will discuss some special techniques that enable affine arithmetic

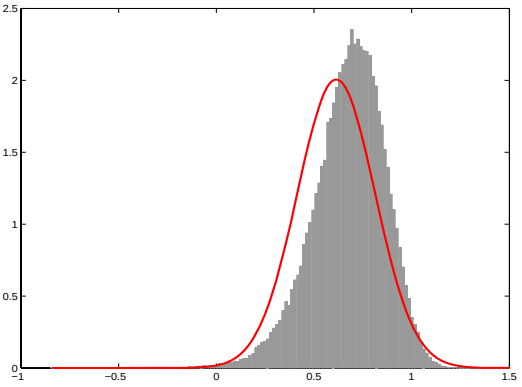


(a) $z = x \cdot y$

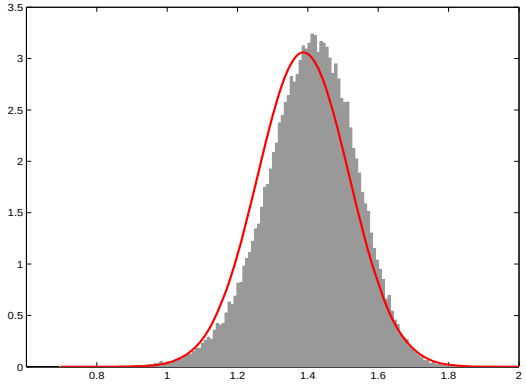
(b) $z = x/y$



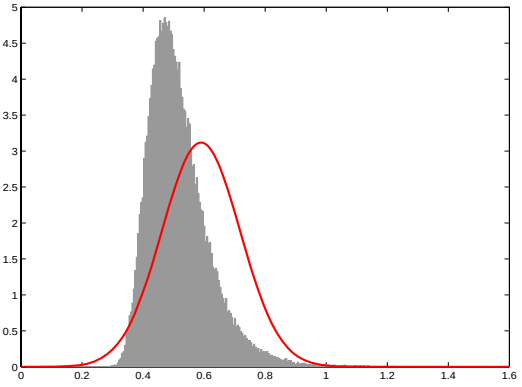
(c) $z = \exp x$



(d) $z = \log x$



(e) $z = \sqrt{x}$



(f) $z = 1/x$

Figure 5.1: The distributions produced from interval analysis for the non-affine functions

The figures plot the probability density functions of z. The histogram is the simulation result (normalized such that the total area is 1), and the curve is produced from the result of interval analysis. In this example, $\hat{x} = 2.0 + 0.25\varepsilon_1 + 0.25\varepsilon_2$, and $y = 2.5 + 0.2\varepsilon_1 + 0.4\varepsilon_2$.

to more accurately represent and propagate probability distributions. We will first illustrate these techniques using a single non-affine function, including $z = x \cdot y$, $z = x/y$, $z = \exp(x)$, $z = \log(x)$, $z = \sqrt{x}$, and $z = \frac{1}{x}$, and then show how we estimate probability distributions with these techniques in some practical nonlinear applications.

5.2 Key Techniques

5.2.1 Representing input distributions with intervals

The first step towards propagating distributions is to represent *input* distributions with affine intervals. In its classical derivation, an affine interval is constructed as an linear combination of uncertainties with finite support, i.e., ε_i is in [-1, 1], and therefore is well suitable to represent the uniform distribution. However, input distributions in realistic applications often do not have finite support, e.g., the normal distribution, and sometimes, they are not even symmetric, e.g., the lognormal distribution. Can we extend the definition of affine intervals so that they are able to model distributions other than the uniform distribution?

The normal distribution is a very common distribution. With a simple modification to the classical definition of affine intervals, we can accurately represent it with an affine form. Here, we define each uncertainty symbol ε_i as a random variable with the standard normal distribution $\mathcal{N}(0, 1)$ (similar to the models in [57,58], as opposed to a bounded range [-1, 1] in its classical definition. Since a linear combination of normal distributions is still a normal distribution, we can represent a normally distributed input with a linear combination of such ε_i's. For example, a random variable x with the $\mathcal{N}(10, 5)$ distribution is represented by $\hat{x} = 10 + 5\varepsilon_1$.

For any other symmetric unimodal distribution, i.e., a symmetric distributions with a single "peak", we need to first approximate it as the normal distribution, and then model it with an affine interval. For an *asymmetric* unimodal distribution, we can model it with an affine form with asymmetric enforced bounds. Suppose the input variable is x, with the mode (the "peak" of the distribution) at x_0. We first set the central value of the affine interval to be x_0, and then sample the

distribution to estimate the *left* and *right* standard deviations,

$$\sigma_l = \sqrt{\sum_i (x_i{}^l - x_0)^2}$$
$$\sigma_r = \sqrt{\sum_i (x_i{}^r - x_0)^2},$$

where $x_i{}^l$'s are all the samples less than x_0, and $x_i{}^r$'s are all the samples greater than x_0. To capture the asymmetry of the distribution, we enforce the upper and lower bounds to be three standard deviations beyond the mode (left and right standard deviations are different):

$$x_l = x_0 - 3\sigma_l$$
$$x_h = x_0 + 3\sigma_r.$$

The affine interval for the variable is $x_0 + \max(x_l, x_h)\varepsilon_1$, where ε_1 has the $\mathcal{N}(0,1)$ distribution. This affine interval, together with the enforced bounds x_l and x_h, approximately models the asymmetric distributed random variable x.

If there are multiple inputs and their correlation is characterized by a covariance matrix, then we can use PCA to find out a set of correlated affine intervals to represent the inputs. Details on PCA have been provided in Section 3.2.3.

However, we note that modeling input distributions with intervals has limitations. If the input distribution is multimodal, it is not possible to capture the multiple "peaks" in the distribution with our affine interval, and therefore it is not recommended to use interval techniques in that scenario.

In the rest of this section, we introduce three key techniques that help to improve the accuracy of distribution analysis. They include *center adjustment*, *asymmetric PDF generation*, and *PDF curve smoothing*. The first one is employed in every interval computation, and the last two are conducted at the end of the entire interval analysis. We will show the improvements by each technique on the six non-affine functions. Since the normal distribution is the most commonly used distribution, we illustrate our techniques on examples where the input distributions are normal.

5.2.2 Center adjustment

One obvious problem with the current interval techniques is that the mode of the estimated PDF, which is the central value of the affine interval, does not match the mode of the actual distribution. Therefore, we need to shift the central value of the resulting affine interval in order to fix this problem.

Suppose the arithmetic operator is $z = f(x)$ or $z = f(x, y)$. Through experiments, we observe that the mode of z is always very close to the nominal value $f(x_0)$ or $f(x_0, y_0)$. However, if we follow the interval computation algorithms (i.e., the minimax approximation for unary operations in Section 4.2.2 and the minivolume approximation for binary operations in Section 4.2.3) to compute the resulting asymmetric affine interval $\hat{z}$, the central value z_0 is usually not at the nominal value. Let us use the exp() function as an example. Suppose the input is $\hat{x} = 2.0 + 0.25\varepsilon_1 + 0.25\varepsilon_2$, where ε_i's have the standard normal distributions. If we use the interval computation algorithm outlined in Figure 4.7 in Section 4.2, the central value for the output is 9.6. However, the nominal value of z is $e^{x_0} = 7.39$, which we found empirically very close to the "peak" of the actual distribution.

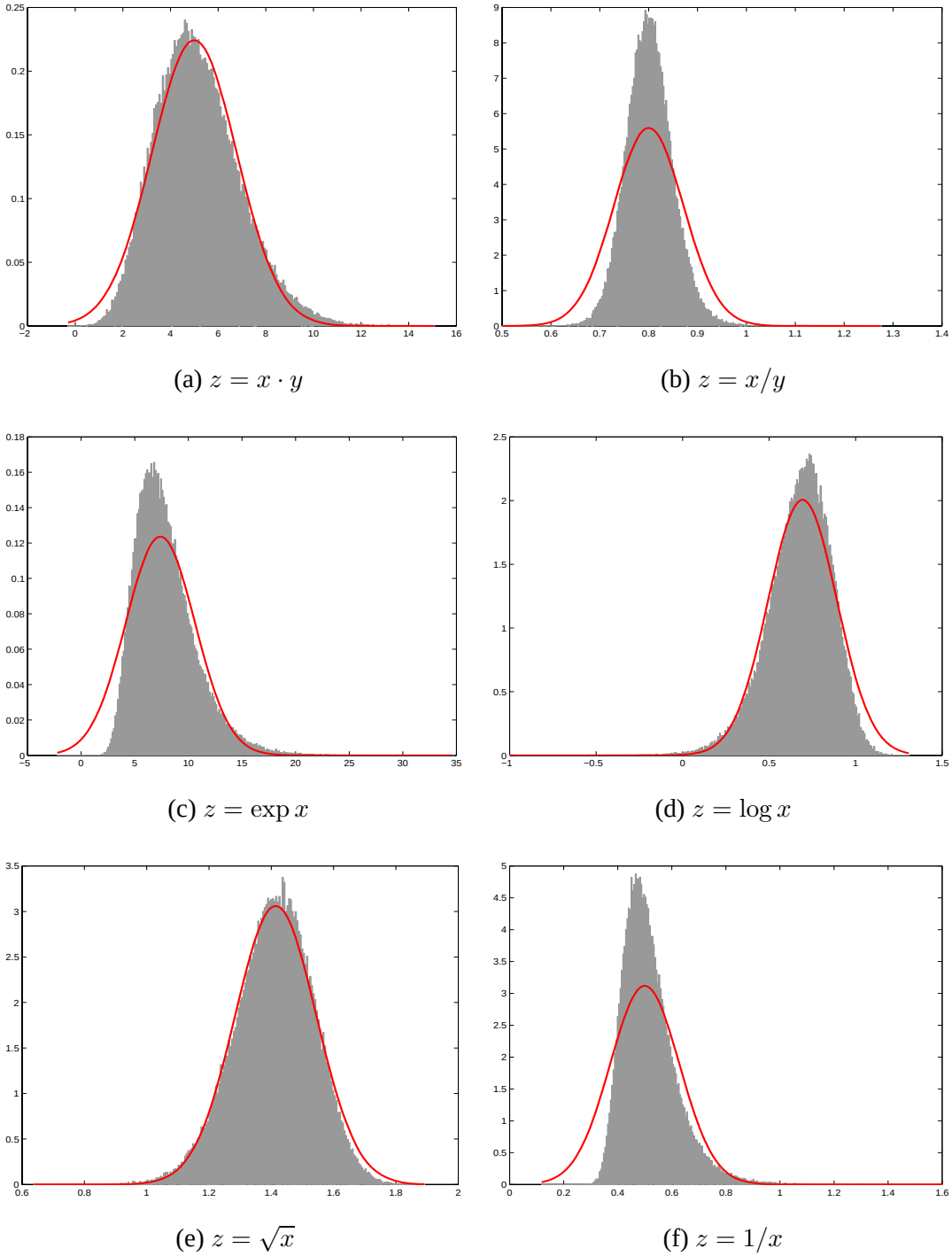
Hence, our center adjustment technique is, for each interval computation, to shift the central value to the nominal value, that is

$$z_0 = f(x_0) \text{ for unary functions, or}$$
$$z_0 = f(x_0, y_0) \text{ for binary functions.}$$

We show the results after center adjustment for the six non-affine function in Figure 5.2. Now, the mode of the estimated PDF fits the mode of the actual distribution very well. However, this treatment does not fix the whole problem: the center-symmetric normal PDF curves still can not capture the skewness of the actual distributions. Hence, we introduce our second empirical enhancement — asymmetric PDF generation.

5.2.3 Asymmetric PDF generation

In the previous chapter, we mitigate the symmetry problem by enforcing asymmetric bounds. In the new problem of interval-based distribution analysis, the enforced bounds can actually be very



(a) $z = x \cdot y$ (b) $z = x/y$

(c) $z = \exp x$ (d) $z = \log x$

(e) $z = \sqrt{x}$ (f) $z = 1/x$

Figure 5.2: The distributions produced from interval analysis after center adjustment

The figures plot the probability density functions of z. The histogram is the simulation result (normalized such that the total area is 1), and the curve is produced from the result of interval analysis. After center adjustment, the mode of the estimated PDF fits the mode of the actual distribution quite well. In this example, $\hat{x} = 2.0 + 0.25\varepsilon_1 + 0.25\varepsilon_2$, and $y = 2.5 + 0.2\varepsilon_1 + 0.4\varepsilon_2$.

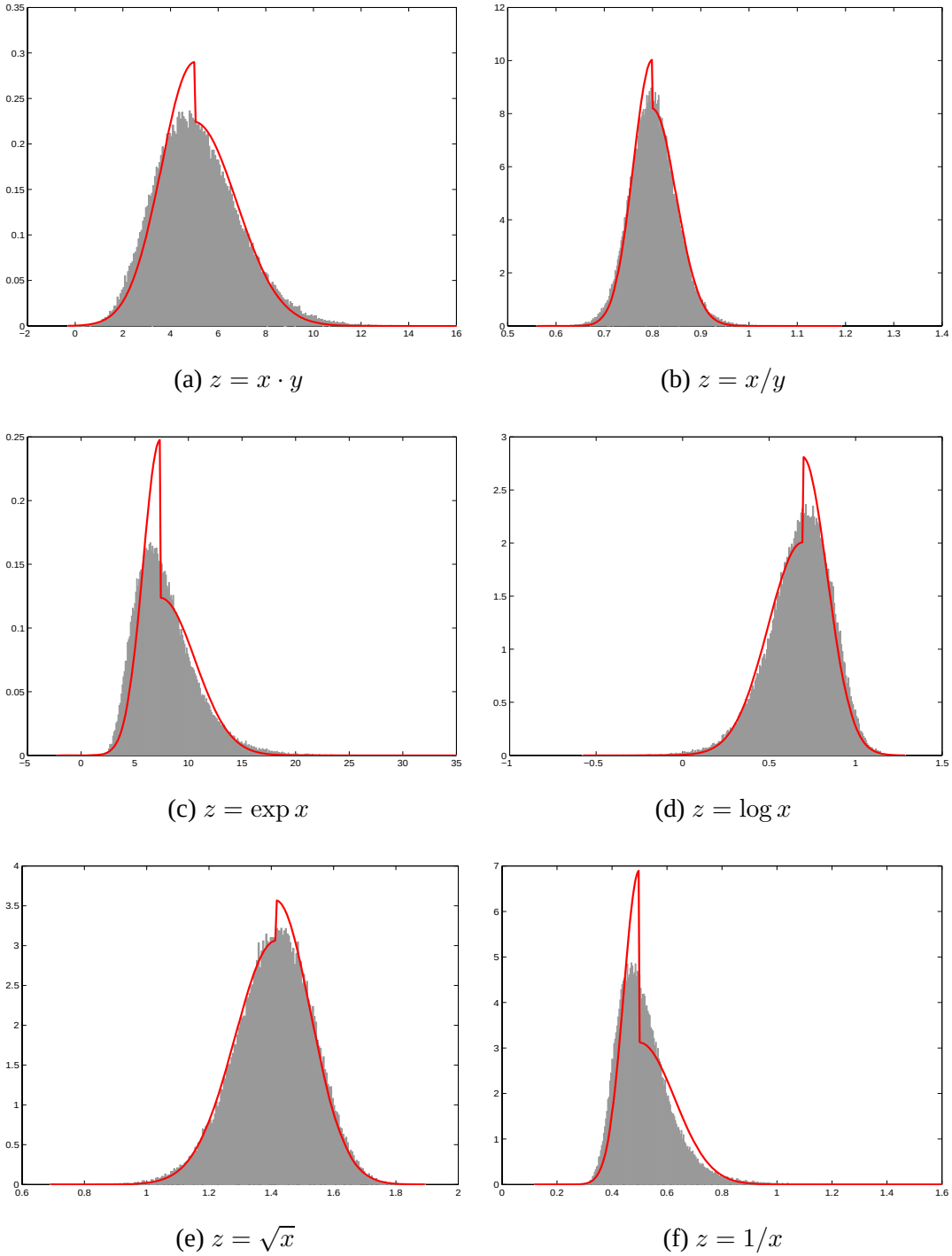
helpful in estimating asymmetric distributions. Recall that due to the approximations made in non-affine interval functions, the bounds implied by the resulting classical affine form are pessimistic and always symmetric to the center, whereas the enforced bounds are more accurate and can be center-asymmetric. If we associate a probability distribution with an asymmetric affine interval, the enforced bounds indicate where "almost all of the" data end at the two sides of the distribution curve. We can use this information to approximate an asymmetric distribution curve. It consists of two *half normal* curves, a left curve and a right curve, with different standard deviations, and the $+3\sigma_r$ point matches the enforced upper bound and the $-3\sigma_l$ point matches the enforced lower bound. Suppose the resulting asymmetric affine interval is $\{z_0 + \sum_{i=1}^{N} z_i\varepsilon_i, z_l, z_h\}$, where z_l is the lower bound, and z_h is the upper bound. The standard deviations for the two half curves are computed as

$$\sigma_l = (z_0 - z_l)/3$$
$$\sigma_r = (z_r - z_0)/3.$$

The PDF is then obtained by directly combining these two asymmetric half curves.

Returning to our previous example, where $\hat{z} = \exp(\hat{x})$ and $\hat{x} = 2.0 + 0.25\varepsilon_1 + 0.25\varepsilon_2$, the lower and upper bounds of z are computed to be 2.56 and 21.34 using the algorithm in Figure 4.7 in Section 4.2, which fit the simulated results very well (see Figure 5.3(c)). In order to generate an asymmetric PDF curve, we compute the left and right standard deviations from these asymmetric bounds, and the results are $\sigma_l = (7.39 - 2.56)/3 = 1.61$ and $\sigma_r = (21.34 - 7.39)/3 = 4.65$, which are then used to generate the two half normal curves.

The improvements by this technique on all of the six non-affine functions are shown in Figure 5.3. Now, both the mode and the two tails of the estimated PDF better match those of the actual distribution. However, directly combining the two half normal curves introduces another problem, i.e., a discontinuous point at the intersection of the two curves. This can be be explained by computing



(a) $z = x \cdot y$ (b) $z = x/y$

(c) $z = \exp x$ (d) $z = \log x$

(e) $z = \sqrt{x}$ (f) $z = 1/x$

Figure 5.3: The distributions produced from interval analysis by utilizing asymmetric bounds

The figures plot the probability density function of z. The histogram is the simulation result (normalized such that the total area is 1), and the curve is produced from the result of interval analysis. By using the asymmetric bounds to produce two half normal curves, we match the tails of the estimated PDF to that of the actual distribution. In this example, $\hat{x} = 2.0 + 0.25\varepsilon_1 + 0.25\varepsilon_2$, and $y = 2.5 + 0.2\varepsilon_1 + 0.4\varepsilon_2$.

the probability density at the point z_0 using the two different standard deviations:

$$\begin{aligned} P(z_0^-) &= \frac{1}{\sqrt{2\pi}\sigma_l} e^{-(z_0 - z_0)^2/2\sigma_l^2} \\ &= \frac{1}{\sqrt{2\pi}\sigma_l} \\ P(z_0^+) &= \frac{1}{\sqrt{2\pi}\sigma_r} e^{-(z_0 - z_0)^2/2\sigma_r^2} \\ &= \frac{1}{\sqrt{2\pi}\sigma_r}. \end{aligned}$$

We see that the two half curves have different probability densities at the intersecting point z_0. The ratio between the two probability densities is

$$\frac{P(z_0^-)}{P(z_0^+)} = \frac{\sigma_r}{\sigma_l}$$

So the more asymmetric the distribution is, the sharper the discontinuity becomes. As Figure 5.3 shows, $z = exp(x)$, $z = log(x)$ and $z = 1/x$ have very skewed distributions, and hence the discontinuities in these curves are very sharp. Next, we will perform curve smoothing to remove the discontinuous peak.

5.2.4 PDF curve smoothing

Smoothing is a process by which data points are averaged with their neighbors [91]. This usually has the effect of blurring the sharp edges. Since the PDF curve generated from an asymmetric affine interval is discontinuous at the center point, we apply local curve smoothing around the center to obtain a smooth PDF curve. This is a simple, empirical heuristic, but it seems to work well.

An important choice in curve smoothing is the *kernel function* which defines the shape of the weight function used to take the average of the neighboring points. Uniform, triangle, and Gaussian functions are common kernels (shown in Figure 5.4). The uniform kernel has the effect of replacing each data point with a straight average of itself and the neighboring points within a certain window. Although it is mathematically the simplest smoothing method, it usually produces a rougher curve because of the abrupt cutoff at the edges of the smoothing window. The other two kernels, however, weight the data points according to their distance to the kernel center, with more weights on the center point. Therefore, by avoiding the abrupt cutoff in the smoothing window, they produce

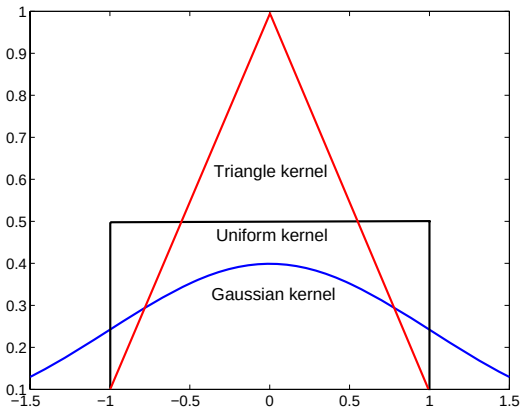


Figure 5.4: Kernel functions

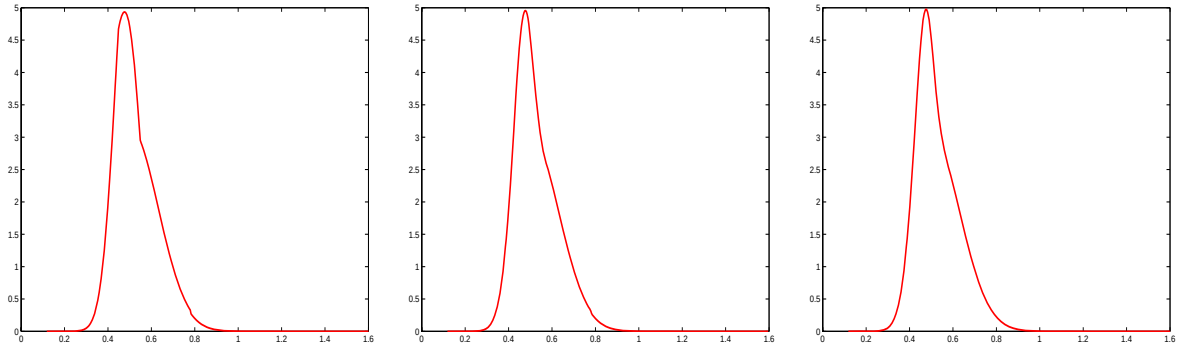
The bandwidth for the uniform kernel, the bandwidth for the triangle kernel, and the standard deviation for the Gaussian kernel are all equal to 1.

empirically smoother curves. In fact, the Gaussian kernel does not have a bounded smoothing window: all the data points contribute to the weighted average, with diminishing weights toward the two ends. So, Gaussian kernel smoothing is the most computationally expensive among the three.

Besides the kernel function, the smoothing *bandwidth* is also a critical parameter in curve smoothing. For the uniform and the triangle kernel, it is the width of the smoothing window, and for the Gaussian kernel, it is the standard deviation of the function. It determines how many neighboring points are averaged to obtained the value of one point, and hence controls the curve's smoothness. Bandwidth selection bears the danger of under- or over-smoothing. Too narrow a window does not smooth sharp edges, and too wide a window averages out useful information.

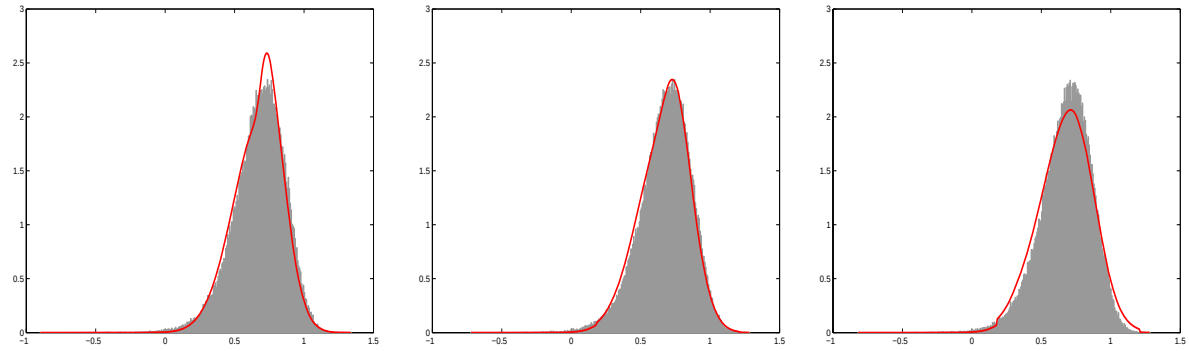
We first study the effects of different kernel functions by applying local smoothing on the PDF curves of our six non-affine function samples. The results for the $1/x$ function are shown in Figure 5.5. Interestingly, the degrees of smoothness from these three kernels are comparable, with the uniform smoothing being slightly worse than the other two. This is mainly because in our application, the curve before smoothing has only one sharp edge, and therefore, it does not require a sophisticated kernel function to smooth out the edge. Due to the computational disadvantage of the Gaussian kernel smoothing, we therefore choose the triangle kernel for our PDF curve smoothing.

The smoothing bandwidth is also chosen through an empirical study. We define the *total width*



(a) Uniform smoothing (b) Triangle smoothing (c) Gaussian smoothing

Figure 5.5: Effects of different smoothing kernels on the PDF of the $1/x$ function



(a) $P = 15\%$ (under-smoothing) (b) $P = 25\%$ (c) $P = 45\%$(over-smoothing)

Figure 5.6: Effects of different smoothing bandwidths on the PDF of the $\log(x)$ function

The smoothing bandwidth P is defined as the ratio between the width of the smoothing window and the total width of a PDF curve.

of a PDF curve as the number of data points between $[z_0 - 3\sigma_l, z_0 + 3\sigma_r]$, and the smoothing bandwidth as the percentage (P) of the total curve width. On the six non-affine functions, we have tried different P's, and find that choosing a smoothing bandwidth around 25% of the total curve width produces the best result. Figure 5.6 shows the results for different bandwidths on the $\log(x)$ function. By comparing to the actual distribution (the histogram in the figure), we can see that narrower ($P = 15\%$) or wider ($P = 45\%$) bandwidth results in slight under- or over-smoothing.

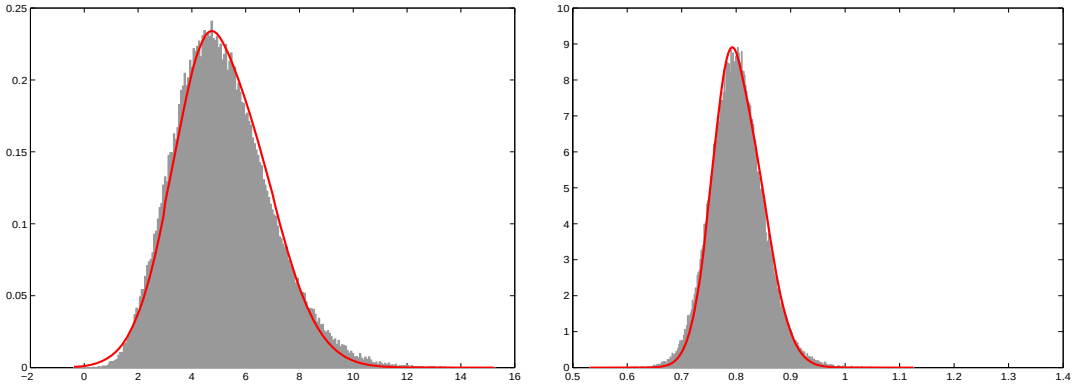
In Figure 5.7, we show the results for the six non-affine function samples again, after applying all of the three techniques, including center adjustment, asymmetric PDF generation, and PDF curve smoothing. Now, the estimated PDF curves match the actual distribution very closely. It is worth mentioning that performing our distribution analysis does not incur much extra computational cost compared to interval analysis using asymmetric affine arithmetic. The representation forms that are propagated through the target application are still asymmetric affine intervals, and do not have any notion of distributions. The entire process has the same computational cost as regular affine interval computations except that at the end, a PDF curve is generated based on the asymmetric bounds and then smoothed around the center of the curve.

Finally, we show the entire process of PDF estimation in Figure 5.8, which summarizes all the techniques we discussed in this section, where they come into play in the process, and the output form after each step. Note that the PDF is generated only after all interval computations are finished. In other words, for all the interval computations, the inputs and the output are still asymmetric affine intervals.

5.3 Experimental Results—Distribution Analysis

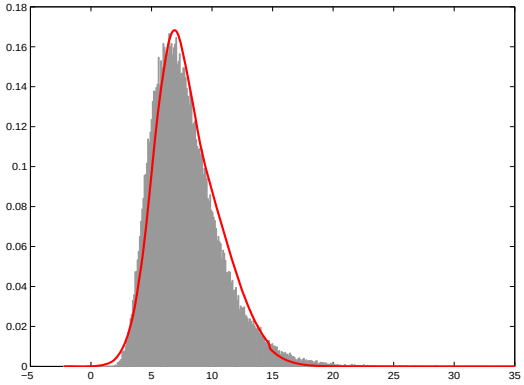
By employing the three key techniques, we can empirically approximate the probability distributions, for the elementary interval computations, using asymmetric affine arithmetic. In this section, we conduct experiments to test the applicability and accuracy of our method on more complex applications.

The central theme of our chosen applications is that they rely on the need to calculate the maximum of a set of uncertainties. This is attractive for several reasons. First, we have already seen

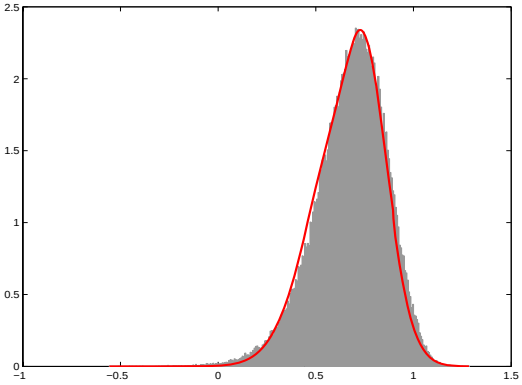


(a) $z = x \cdot y$

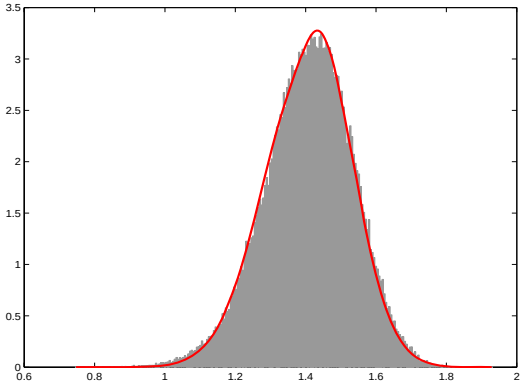
(b) $z = x/y$



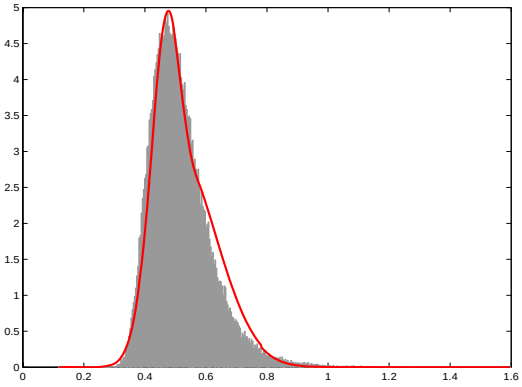
(c) $z = \exp x$



(d) $z = \log x$



(e) $z = \sqrt{x}$



(f) $z = 1/x$

Figure 5.7: The distributions produced from interval analysis after curve smoothing

The figures plot the probability density function of z. The histogram is the simulation result (normalized such that the total area is 1), and the curve is produced from the result of interval analysis. In this example, $\hat{x} = 2.0 + 0.25\varepsilon_1 + 0.25\varepsilon_2$, and $y = 2.5 + 0.2\varepsilon_1 + 0.4\varepsilon_2$.

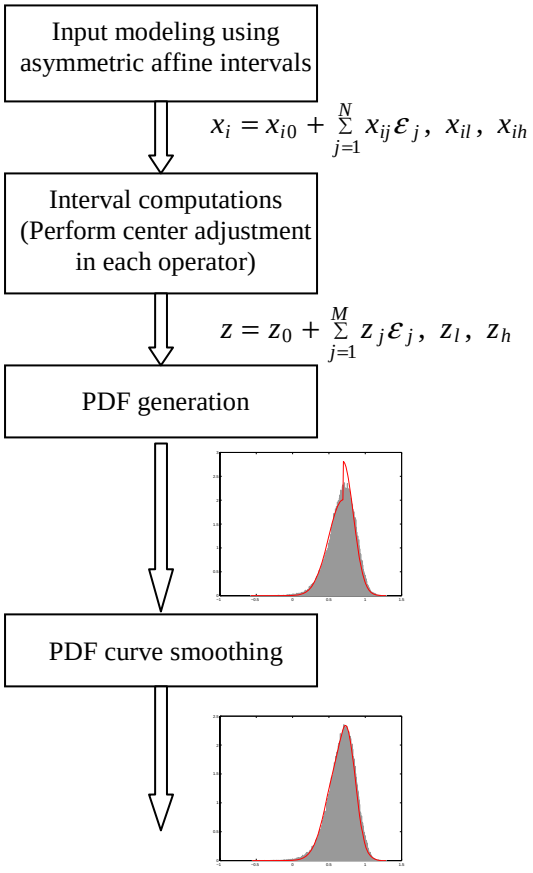


Figure 5.8: Flow chart of PDF estimation

This flow chart summarizes all the techniques we discussed in this section, where they come into play in the process, and the output form after each step. In the figure, x_i's are the inputs of an application and z is the output.

the central role that calculating max/min play in successful statistical applications such as SSTA. But what is more interesting for our purposes is that the standard nonlinear maximum operator can be approximated in a purely continuous form we refer to as the "soft-max" operator. Soft-max approximates the true maximum, but requires addition, multiplication, division, exp, and log in its computation. Thus, it forms a nice "stress" test for our statistical intervals, and we show a set of synthetic tests with varying numbers of these maximum computations. To analyze a more realistic application, we show how to apply these ideas to a Viterbi decoder that is used in tasks such as Hidden Markov Modeling.

5.3.1 Background: the soft-max approximation

We first offer some background on the soft-max operator. The name '*soft-max*' comes from the fact that it is a smooth, i.e., fully continuous and differentiable, approximation to the max function. A binary soft-max operator is defined as

$$z = \text{smax}(x, y) = \frac{1}{k} \log(e^{kx} + e^{ky}),$$

where k is a constant scaling factor. It approximates the $z = \max(x, y)$ function in the sense that the exponential function amplifies the distance between x and y, and the output z is more influenced by the larger one. It is not a true max operator because the max function returns either $z = x$ or $z = y$ depending on their magnitudes, while the soft-max function returns a value contributed to by both of the inputs. The scaling factor affects the "sharpness", or the proximity to the real max function, of the soft-max function. The higher the k, the sharper this soft-max function is. In fact, the following inequality between the two functions holds [5],

$$\max(x, y) \leq \text{smax}(x, y) \leq \max(x, y) + \frac{\log 2}{k}.$$

When k is small, the output of the soft-max could be much larger than the real maximum, and as k approaches infinity, the soft-max function approaches the real max function.

Next, we use a simple example to further compare the soft-max function with the real max function and illustrate how their proximity is affected by the scaling factor k. In this example, we fix one of the inputs $y = 10$, and study how the output z changes with x. For the real max function, there are two possible outcomes: $z = 10$ and $z = x$, and they intersect at the point $x = 10$ (see

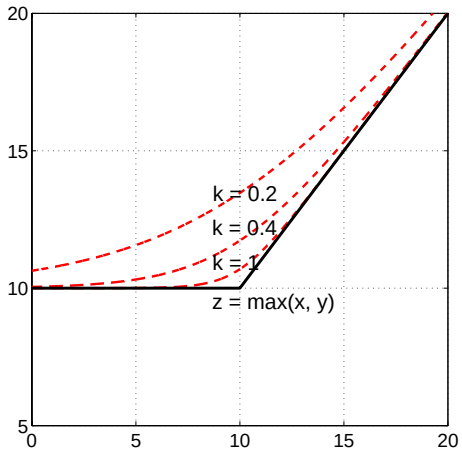


Figure 5.9: Soft-max vs. max

In this example, y is fixed to be 10, the solid curve is for the max function $z = \max(x, 10)$, and the dotted curves are for the soft-max function $z = \text{smax}(x, 10)$, with varying scaling factor k. As k gets larger, the soft-max function approaches the real max function.

Figure 5.9). In contrast, the soft-max function has the effect of smoothing out the corner of the curve for the max function. The distance between the max and the soft-max is affected by two things: one is the difference between the inputs x and y, and the other is the scaling factor k. When k is fixed, the soft-max function is closer to the max function when the two inputs x and y are farther away from each other. When the two inputs are close, the soft-max function becomes "softer", or less like the max function, as k gets smaller.

When there are multiple inputs, the real max function can be implemented recursively by employing the binary max function multiple times, because

$$\max(x_1, x_2, ..., x_n) = \max(\max(x_1, x_2, ..., x_{n-1}), x_n).$$

The same rule proves to be true for the soft-max function as well:

$$\begin{aligned}
\text{smax}(x_1, x_2, ..., x_n) &= \frac{1}{k}\log(e^{kx_1} + e^{kx_2} ... + e^{kx_n}) \\
&= \frac{1}{k}\log(e^{\log(e^{kx_1}+e^{kx_2}...+e^{kx_{n-1}})} + e^{kx_n}) \\
&= \text{smax}(\text{smax}(x_1, x_2, ..., x_{n-1}), x_n).
\end{aligned}$$

Therefore, the soft-max function with N inputs can be implemented by repeatedly using the binary soft-max function $N - 1$ times.

One reason we are interested in the soft-max is that it has a very intriguing application in modeling gate delays in circuit timing analysis. When two input signals pass through a logic gate, the arrival time of the output usually depends on the maximum of the two inputs' arrival times. We explain it with a simple gate delay model. Suppose the arrival times for the two inputs are t_1 and t_2, and the pin-to-pin delays are d_1 and d_2 for the two paths from the inputs to the output. Then the arrival time of the output is usually modeled as

$$t_o = \max(t_1 + d_1, t_2 + d_2).$$

However, this model is accurate only for the case of single input switching. When multiple inputs of a gate switch in temporal proximity, the maximum delay model tends to underestimate the output arrival time. The effects of temporal proximity of input transition on gate delay have been discussed in [41, 68], and a quite complex proximity model is introduced in [12] in the context of static timing analysis (STA). Interestingly, our preliminary experiments show that gate propagation delay can actually be modeled rather well with the simple soft-max function, for both single input switching and multiple input switching,

$$t_o = \text{smax}(t_1 + d_1, t_2 + d_2).$$

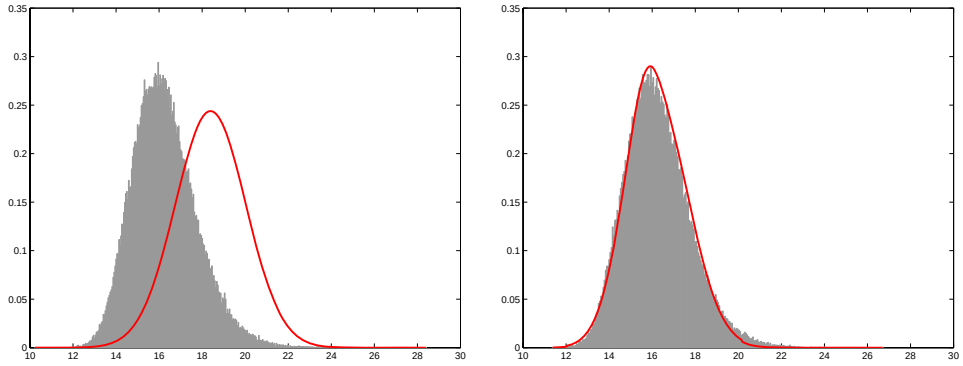
The experimental results on modeling gate delay with the soft-max function are presented in Appendix E, as it is not the main focus of this thesis.

However, in this chapter, we limit ourselves to the problem of understanding how well we can model the chain of nonlinear computations involving soft-max with our statistical interpretation of intervals, and how well an established DSP task such as Viterbi decoding (which relies on the max operator) can be analyzed.

5.3.2 A single soft-max operator

Our first simple experiment is to test how our techniques perform on a single soft-max operator which is composed of six nonlinear functions and an addition. In the experiment, the two inputs x and y are normally distributed, and are modeled as the following affine intervals:

$$\begin{aligned} x &= 2.0 + 0.25\varepsilon_1 + 0.25\varepsilon_2 \\ y &= 2.5 + 0.2\varepsilon_1 + 0.4\varepsilon_2 \end{aligned} \tag{5.1}$$



(a) Before (b) After

Figure 5.10: Distribution analysis on a single soft-max operator

The histogram is from simulation and the curve is generated by our distribution analysis. (a) and (b) show the result before and after applying the three key techniques introduced in this chapter.

The central values and the variations are chosen such that the distributions of x and y are well overlapped, which is a "stress" case for the soft-max operator.[1] The scaling factor k is now also a normal random variable, independent of x and y, and is modeled as

$$k = 0.05 + 0.005\varepsilon_3 \tag{5.2}$$

We conduct distribution analysis on a single soft-max operator with and without the three techniques introduced in this chapter. The results are compared in Figure 5.10. The histogram is from Monte Carlo simulation with 10^5 samples. We can see that without the special treatments, affine arithmetic captures the bounds, but overestimates the mean value and fails to capture the skewness of the actual distribution. As a comparison, the PDF estimated by the new interval techniques matches the actual distribution very well. We measure the estimation accuracy using the relative errors of the 1%, 10%, 25%, 50%, 75%, 90%, and 99% points of the distribution, compared to the simulation results. The accuracy measures are shown in Table 5.1. The relative errors of these percentiles are less than 1.5%.

However, as we increase the mean value of k, or increase the "sharpness" of the soft-max function, the estimation accuracy drops. Figure 5.11(a) shows that our distribution analysis significantly

[1] When the distributions of x and y are well apart, the output of the soft-max operator is influenced largely by only one of the inputs, and hence, propagating normal distributions from the inputs to the output is relatively easy. For example, when y is mostly larger than x, $\frac{1}{k}\log(e^{kx} + e^{ky}) \approx \frac{1}{k}\log(e^{ky}) = y$.

Percentile	Relative error
1% point	1.22%
10% point	0.37%
25% point	0.21%
50% point	0.01%
75% point	0.04%
90% point	0.89%
99% point	1.42%

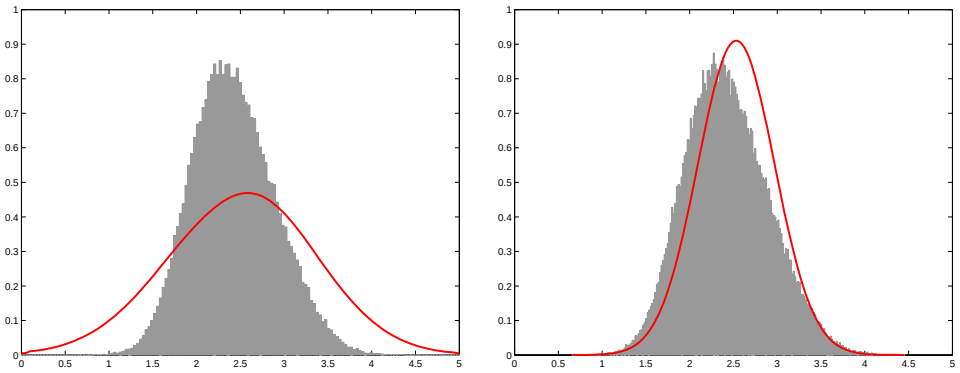
Table 5.1: Estimation error for the distribution analysis on a soft-max operator (k = 0.05)

overestimates the standard deviation when $k = 1$. The main reason for the accuracy degradation is that the exp() function always amplifies the estimation error in the previous computation, i.e., the multiplication kx or ky, and the larger the k, the more amplification effect it has. Fortunately, there is a simple trick that can alleviate this problem. The soft-max function can be rearranged as the following:

$$\begin{aligned} \text{smax}(x, y) &= \frac{1}{k}\log(e^{kx} + e^{ky}) \\ &= \frac{1}{k}\log(e^{kx}(1 + e^{k(y-x)})) \\ &= \frac{1}{k}(kx + \log(1 + e^{k(y-x)}) \\ &= x + \frac{1}{k}\log(1 + e^{k(y-x)}) \end{aligned} \tag{5.3}$$

This alternative implementation has two advantages. First, it has only one exp() function instead of two, reducing the chance of error amplification. Second, the argument of the exp() function now becomes $k(y - x)$ which is usually smaller than kx or ky (provided that x and y are close), and as a result, the effect of error amplification is also smaller. The result for $k = 1$ using the alternative implementation is shown in Figure 5.11(b). Compared to the default implementation, the new implementation does move the estimated PDF much closer to the actual distribution, although it is not as accurate as the case for a smaller k. The relative errors of the percentiles range from 0.51% to 3.81%, shown in Table 5.2.

Next, we study how the variations of x, y and k in the soft-max function affect the accuracy of our distribution analysis. Note that since k appears in the denominator in the soft-max function,



(a) Default implementation (b) Alternative implementation

Figure 5.11: Performance on a soft-max operator with larger k

The histogram is from simulation and the curve is generated by our distribution analysis. This example has a scaling factor $k = 1$. The default implementation (a) overestimates the standard deviation as k gets larger, while the alternative implementation (b) produces a much more accurate PDF.

Percentile	Relative error
1% point	3.21%
10% point	3.35%
25% point	2.78%
50% point	1.29%
75% point	0.51%
90% point	2.31%
99% point	3.81%

Table 5.2: Estimation error for the distribution analysis on a soft-max operator (k = 1)

		Variations of the inputs x and y	
		Small (18%)	Large (72%)
Variation of k	Small (10%)	case A	case B
	Large (25%)	case C	case D

Table 5.3: Four different variation settings

The percentage is the ratio between the standard deviation and the mean value.

the variation of k has to be constrained so that the lower bound of k (which is regarded as the -3σ point) does not include zero. More specifically, k's standard deviation should not exceed 33% of it mean. We chose 10% and 25% as the low and high variation settings for k (the percentage is the ratio between the standard deviation and the mean value). In case of 25% variation, k is modeled as

$$k = 0.05 + 0.0125\varepsilon_3.$$

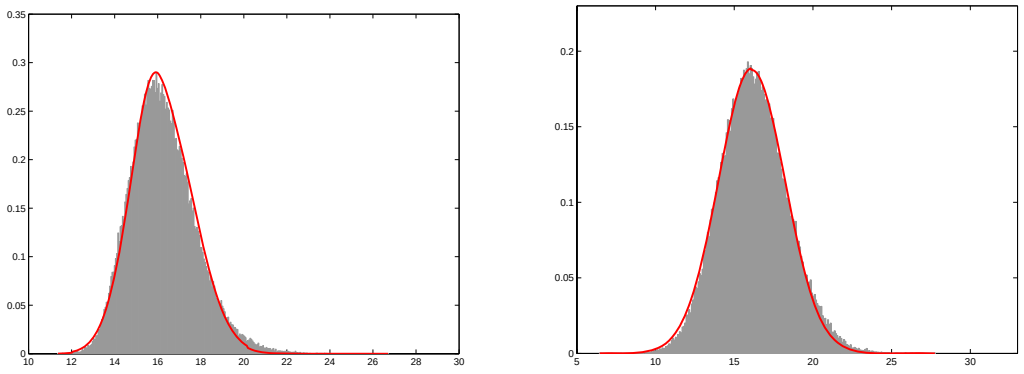
For the inputs x and y, currently they both have 18% variations. We chose 72% as the high variation setting in which case the inputs are modeled as

$$\begin{aligned} x &= 2.0 + 1.0\varepsilon_1 + 1.0\varepsilon_2 \\ y &= 2.5 + 0.8\varepsilon_1 + 1.6\varepsilon_2 \end{aligned} \tag{5.4}$$

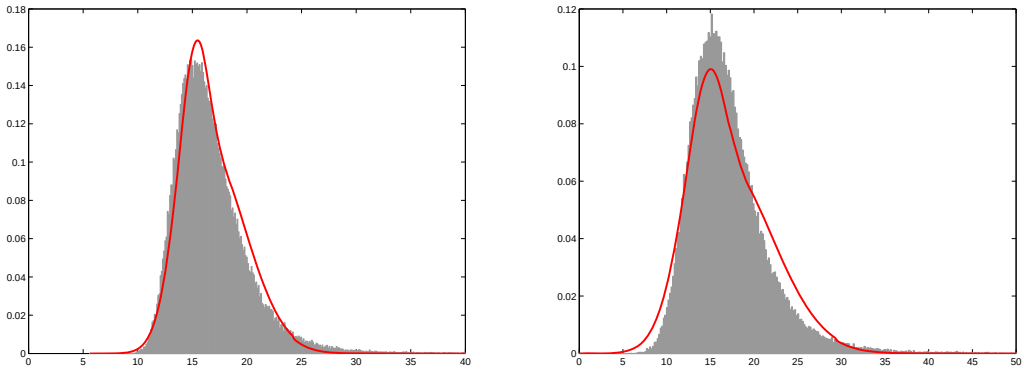
There are four different variation combinations, as listed in Table 5.3. The estimated distributions for these four cases are shown in Figure 5.12, and their accuracy measures are compared in Table 5.4. By comparing the result for case A with that for case B, we observe that the accuracy is not very sensitive to the variations of the inputs: it still produces a fairly accurate PDF even with 72% input variations, given the scaling factor variation is small (10% variation). However, when the variation in k increases to 25%, the accuracy noticeably falls: in case D, the relative error for 1% point reaches 12%.

5.3.3 Binary tree with soft-max operators

The results of the previous section give a good picture of how the soft-max operator can be computed using our statistical interval approximation, and what some of the accuracy issues are. But these results are really more for purposes of illustration. Hence, in this section, we look at a larger, albeit synthetic problem, to explore one of the really serious concerns with all interval-valued computation: the problem of loss of accuracy as we push intervals through deeper chains of computation. In the case of range-only uncertainty, i.e., without any explicit statistical interpretation, the problem is range explosion. Now, we seek to understand what happens to our heuristic for approximating the underlying PDF in the same context. To do this, we stay with our soft-max operator, but now deploy it in large, variable depth binary trees. We can control the number and depth of the computations with this synthetic test, which makes this an attractive test case. As shown in Figure 5.13, for an



(a) Case A (b) Case B



(c) Case C (D) Case D

Figure 5.12: Distribution analysis on a single soft-max function under four different variation settings

The histogram is from simulation and the curve is generated by our distribution analysis. The variation settings for the four cases are listed in Table 5.3.

	Relative error			
Percentile	Case A	Case B	Case C	Case D
1% point	1.22%	1.85%	2.12%	12.16%
10% point	0.37%	0.95%	1.20%	3.80%
25% point	0.21%	0.57%	0.92%	1.40%
50% point	0.01%	0.52%	0.83%	1.28%
75% point	0.04%	0.96%	3.47%	5.05%
90% point	0.89%	0.92%	3.86%	6.75%
99% point	1.42%	2.12%	6.82%	8.95%

Table 5.4: Estimation error comparison for four different variation settings

The variation settings for the four cases are listed in Table 5.3.

N-layer binary tree, there are $2^N - 1$ soft-max operators, the longest computational chain is of depth N, and the final output is a function of all 2^N inputs.

In this section, we experiment on a N-layer binary tree with the inputs similar to the ones in the previous section. Two input variation settings are tested. In case of small input variation (18%), the inputs are modeled as

$$\begin{aligned} x_i &= 2.0 + 0.25\varepsilon_i + 0.25\varepsilon_{i+1} \\ x_{i+1} &= 2.5 + 0.2\varepsilon_i + 0.4\varepsilon_{i+1}. \end{aligned} \tag{5.5}$$

where $i = 1, 3, 5, ...$ In case of large input variation (72%), the inputs are

$$\begin{aligned} x_i &= 2.0 + 1.0\varepsilon_i + 1.0\varepsilon_{i+1} \\ x_{i+1} &= 2.5 + 0.8\varepsilon_i + 1.6\varepsilon_{i+1}. \end{aligned} \tag{5.6}$$

We have shown in the previous section that our distribution analysis yields accurate estimates for a single soft-max operator only when the scaling factor k has a small variation, and therefore, in this experiment, we assign 10% variation to k. The affine form representation of k is

$$k = 0.05 + 0.005\varepsilon_k,$$

where the noise symbol ε_k is independent of other noise symbols in the inputs. We test the two cases that have different input variation settings on three binary trees with depths equal to 4, 8 and 12.

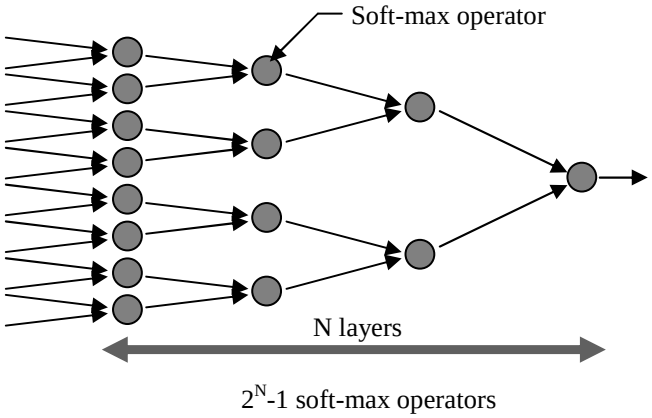


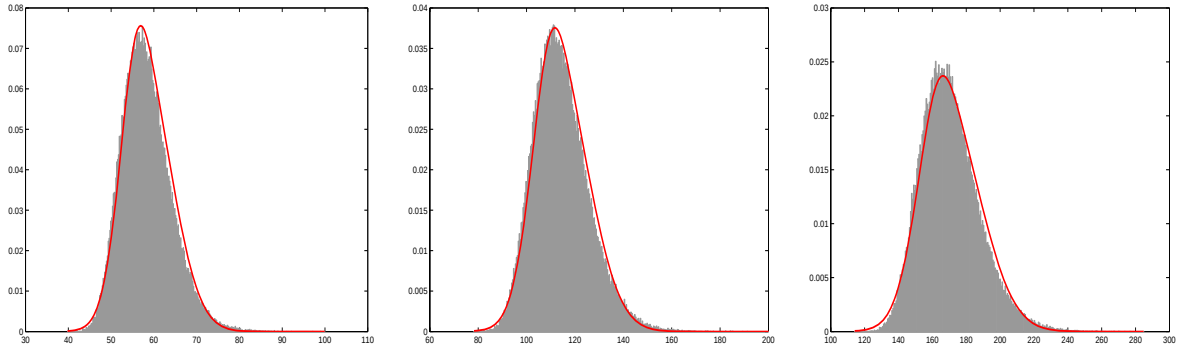
Figure 5.13: A binary tree of the soft-max operators

	Small input variation (18%)			Large input variation (72%)		
Percentile	N=4	N=8	N=12	N=4	N=8	N=12
1% point	0.74%	0.83%	1.25%	0.78%	0.88%	1.44%
10% point	1.01%	1.52%	1.92%	0.42%	1.21%	1.59%
25% point	0.72%	1.08%	1.09%	0.28%	1.15%	1.13%
50% point	0.04%	0.02%	0.68%	0.08%	0.36%	0.35%
75% point	0.31%	0.18%	0.24%	0.35%	0.32%	0.33%
90% point	1.18%	1.19%	1.19%	1.30%	0.94%	1.23%
99% point	2.41%	3.22%	3.71%	2.76%	3.60%	4.19%

Table 5.5: Estimation accuracy (relative errors of percentiles) for binary trees

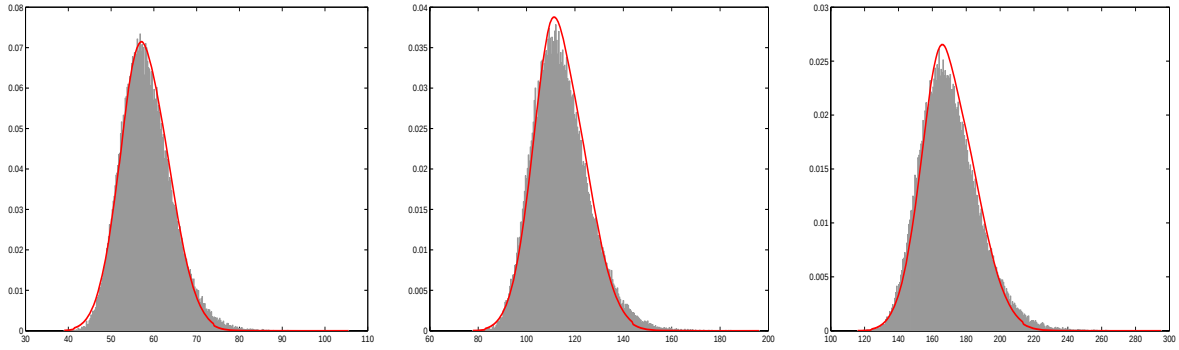
The estimated PDF's are compared to the histograms obtained from simulation in Figure 5.15 and Figure 5.14, and the accuracy measures are compared in Table 5.5. Regardless of the input variation, our distribution analysis provides fairly accurate estimates of PDF's for the three binary trees with increasing complexity. Estimation accuracy slightly drops as the depth of the tree is increased from 4 to 12: the relative error for the 99% point is increased from 2.41% to 3.71% in case of small input variation, and from 2.76% to 4.19% in case of large input variation.

We also realize that our distribution analysis achieves accuracy at the expense of computational complexity. Compared to Monte Carlo simulation with 10^5 samples, the analytical method is about only 20–50X faster, shown in Table 5.6 (see the fifth column). This speedup is not impressive,



(a) N = 4 (b) N=8 (c) N=12

Figure 5.14: Distribution analysis on a soft-max binary tree (input variation = 18%)



(a) N = 4 (b) N=8 (c) N=12

Figure 5.15: Distribution analysis on a soft-max binary tree (input variation = 72%)

The histogram is from simulation and the curve is generated by our distribution analysis. Our distribution analysis provides consistently accurate estimate for soft-max trees with increasing complexity.

N	# of noise symbols	Distribution analysis (sec)	Simulation (sec)	Speedup
4	63	0.1	4.8	48
8	1023	2.6	80	31
12	16383	59.5	1300	22

Table 5.6: Computational cost of the distribution analysis on a soft-max binary tree

Distribution analysis method	Visweswariah [87]	Le [50]	Chang [13]	Ours
Rough speedup	10^4	10^3	10^3	20-50

Table 5.7: Rough speedup comparison of distribution analysis methods

Speedup is measured against Monte Carlo simulation with 10^5 samples.

especially when compared to the state-of-the-art analytical SSTA approaches where the speedup over Monte Carlo simulation is often in the order of 10^3–10^4 [13, 50, 87] (a comparison of rough speedups is shown in Table 5.7). Of course, these comparisons are necessarily qualitative, since our synthetic softmax trees are at best highly simplified synthetic versions of real combinational logic networks used in SSTA. Also, the soft-max computation is an extremely complex way of computing the maximum, even for affine intervals. If our interest was only in efficiency, it is possible to define max(x,y) using operations on the symmetric polygon or ellipse approximation, just as with other binary interval operators. Nevertheless, the comparison does serve the purpose of illustrating the speed/flexibility tradeoffs embodied in interval methods. We clearly pay a price for this flexibility. However, the gain of extra computational cost includes higher accuracy in all interval computations, more ease in handling correlations and better applicability to nonlinear applications. It is also worth noting that our current implementation is focused solely on accuracy, leaving plenty of possibilities for further speedup. One such possibility is heuristic noise symbol management. As we show in the second column of Table 5.6, there are more than 16,000 noise symbols in the output of a 12-layer soft-max binary tree. Propagating and processing such a large number of noise symbols contribute to the slowdown of the distribution analysis. One can use clever heuristics to "condense" the noise symbols without significant degradation in accuracy.

5.3.4 Viterbi trellis with soft-max operators

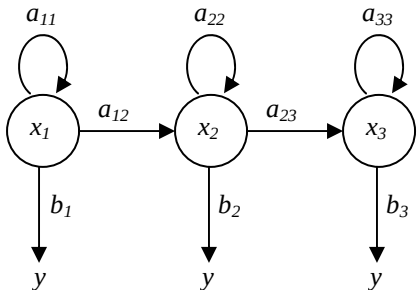
Let us turn our attention now to a more realistic application from the DSP world, the *Viterbi algorithm*, which relies heavily on computing the max of a pair of quantities. It plays an important role in decoding the Hidden Markov Models (HMM) in speech recognition [73]. A HMM is a statistical model where the system being modeled is assumed to be a Markov process, but the states themselves are hidden (or not observable) and each state generates an observation with a certain output probability (see an example in Figure 5.16). In speech recognition, for example, the hidden states may represent words or phonemes, and the observations represent the acoustic signal. The goal of decoding a HMM is to infer the hidden state sequence from the observation sequence. The Viterbi algorithm is a dynamic programming algorithm that finds the most likely sequence of hidden states for HMM. For a given model, let $\phi_j(t)$ represent the maximum likelihood of observing output y_1 to y_t and being in state j at time t. This likelihood can be computed efficiently using the following recursion:

$$\phi_j(t) = \max_i\{\phi_i(t-1)a_{ij}\}b_j(y_t) \tag{5.7}$$

This recursion forms the basis of the Viterbi algorithm. At each recursion step, $\phi_j(t)(j = 1, ..., n)$ is computed from $\phi_j(t-1)(j = 1, ..., n)$, and the computations involve the max operators and multiplications. Finally, at time T, the likelihood of observing the sequence y_1 to y_T equals $\max_i\{\phi_i(T)\}$, and the most likely sequence is determined by tracing from time T back to time 1 and finding the "winning" states of the max operator at each recursion step.

The Viterbi algorithm can be visualized as finding the best path (i.e., the path with the largest likelihood) in a Viterbi trellis (see Figure 5.17). In our experiment, we adopt the Viterbi trellis structure with four states and replace the max operator with the soft-max operator. At each time step of a Viterbi trellis, the signals have to be multiplied by the transition probabilities before going to the soft-max operators. These multiplications add complexity to the algorithm, which makes our distribution analysis more interesting.

To test how our distribution analysis technique performs on a Viterbi trellis, we set the inputs to



x – states of Markov model
y – observable outputs
a – transition probabilities
b – output probabilities

Figure 5.16: A Hidden Markov Model example

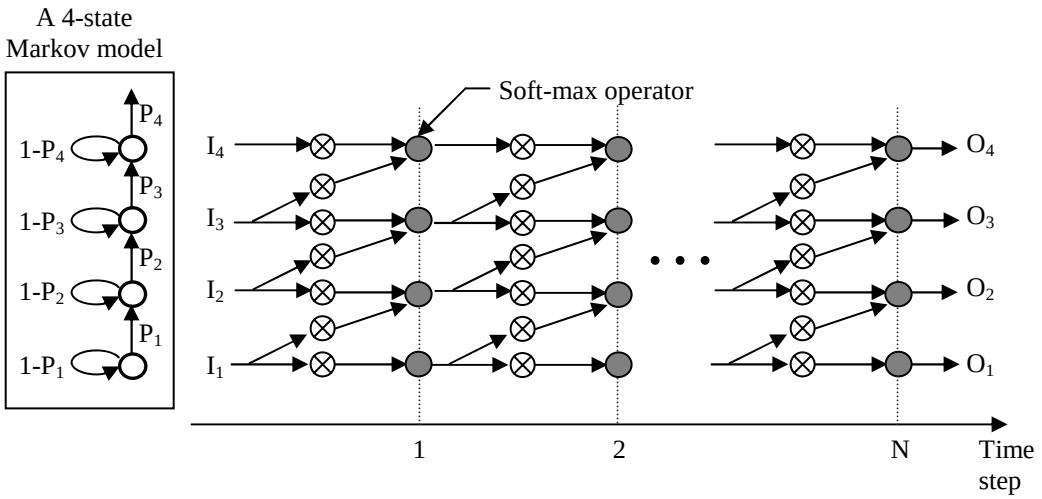
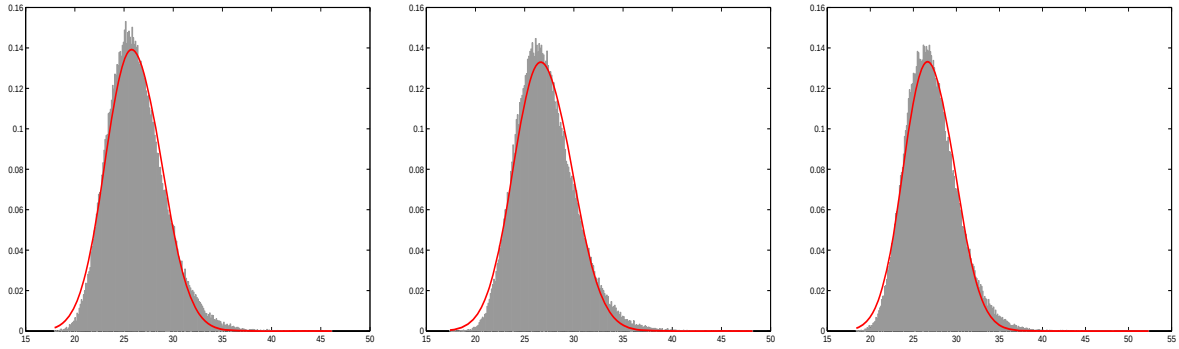


Figure 5.17: The structure of a Viterbi trellis with the soft-max operators



(a) N = 4 (b) N=50 (c) N=100

Figure 5.18: Distribution analysis on a soft-max Viterbi trellis

The histogram is from simulation and the curve is generated by our distribution analysis. Our distribution analysis provides consistently accurate estimate for soft-max Viterbi trellises with increasing complexity.

the trellis as the following correlated affine intervals:

$$I_1 = 2.0 + 0.25\varepsilon_1 + 0.25\varepsilon_2$$

$$I_2 = 2.5 + 0.2\varepsilon_1 + 0.4\varepsilon_2$$

$$I_3 = 2.0 + 0.25\varepsilon_3 + 0.25\varepsilon_4$$

$$I_4 = 2.5 + 0.2\varepsilon_3 + 0.4\varepsilon_4,$$

and the scaling factor of the soft-max function, k, to be

$$k = 0.05 + 0.005\varepsilon_5.$$

The transition probabilities, P_1–P_4, are independent normal random variables with the distribution $\mathcal{N}(0.5, 0.05)$. The objective is to estimate the PDF for the output O_4 at time step N. The estimated PDF's for $N = 4$, 50 and 100 are shown in Figure 5.18, and the accuracy measures are provided in Table 5.8. We see that even with the added multiplications, our distribution still yields accurate results (percentile relative error less than 4.4%), regardless of the trellis complexity.

5.4 Summary

In the previous chapter, we showed how to extend the basic affine interval formula to allow asymmetric uncertainty ranges. In this chapter, starting from this base, we showed how to layer on top

Percentile	N=4	N=50	N = 100
1% point	4.24%	4.33%	4.29%
10% point	0.96%	1.34%	1.46%
25% point	0.02%	0.26%	0.46%
50% point	0.01%	0.02%	0.03%
75% point	0.20%	0.40%	0.40%
90% point	0.00%	0.37%	0.37%
99% point	3.64%	3.58%	4.22%

Table 5.8: Estimation accuracy for Viterbi trellis

of the asymmetric affine interval model a statistical interpretation that allows us to ”retrieve” an approximation of the PDF underlying our interval model. The technique is heuristic, and involves the steps of matching the mode of the final distribution, matching the variance independently on each side of the mode, and then smoothing the overall PDF with a suitable shaped kernel filter. Preliminary results are encouraging: across a set of basic tasks involving the soft-max approximation to the maximum of a pair of variables, we see usable levels of accuracy (percentile relative error less than 4.4%).

However, we note that our approach is only applicable to uncertainties with unimodal distributions. If the subject to study has a multimodal distribution, e.g., wirelength distribution on a chip [89], it is not possible to estimate the PDF using our interval-based approach.

What we have not focused on in this thesis is speed per se; the statistical interpretation of the asymmetric affine interval model adds relatively little overhead, but the asymmetric intervals themselves are complex to propagate through the basic arithmetic and transcendental operators. How fast one might be able to make these computations go, and the extent to which this statistical model is able to compete with, for example, smart Monte Carlo-based approaches, is the subject for further research. However, the results in this chapter do open the door to consideration of affine intervals as a first-order representation for correlated statistical uncertainties in other applications. The work of Ma [57] is an early application of this sort of strategy, albeit with a much simpler statistical interpretation of an affine interval model. We believe these methods should allow others to develop interval-based attacks on other problems.

Another area for future work is to develop a closed-form distribution function. Our current approach is a heuristic, and it provides a look-up table for a smoothed PDF as the end result. In real applications, it may be more desirable to have a closed-form distribution function. It may be possible to use a more analytical method to construct a "best fitting" distribution function based on the PDF look-up table that we obtain from the asymmetric affine interval.

Chapter 6

Conclusions and Future Work

Motivated by ever-increasing uncertainties in DSP and VLSI design, this thesis proposes a novel interval-valued approach for representing and reasoning about uncertainties. It contributes several techniques for improving the accuracy and capability of interval computations. This chapter reviews the dissertation's main contributions and then discusses promising directions for future research.

6.1 Contributions

The application of interval-valued computations has long been hindered by its overly conservative handling of uncertainties. In this dissertation, we enhance the affine interval representation [22] with a probabilistic interpretation and explore the advantages of utilizing probabilistic information during interval computations. The five main contributions of this thesis are summarized as follows:

- **Bring the notion of probabilistic distribution into interval representation**
  Conventionally, intervals and distributions are two completely different representation forms for uncertain quantities. Intervals capture the bounds of uncertainties, and these bounds can be very conservative due to the extremely low likelihood of being reached. In Chapter 3, we provide a probabilistic interpretation for affine intervals based on the Central Limit Theorem and use this interpretation to reduce the pessimism in the bounds. The benefit of this augmentation is even more significant in interval computations. By extending the probabilistic interpretation

to a 2D affine interval, we are able to identify the highly probable areas in the input range of an interval operation. Consequently, we can reduce the pessimism in the interval operation by ignoring the extremely unlikely areas (Chapter 4). Moreover, we discuss how to construct affine intervals from given probabilistic information (Chapter 3). This is especially helpful when the input uncertainties are correlated. These techniques are applied in a DSP application—range and error analysis in finite-precision DSP design.

- **Enable the representation of non-center-symmetric intervals**

  A fundamental limitation of affine arithmetic is that the interval represented by an affine form must be center-symmetric. This restriction highly limits the use of AA in applications where the underlying uncertainties are not symmetric. In Chapter 4, we proposed the *asymmetric affine interval*, which is an affine interval with enforced asymmetric bounds. The enforced bounds serve as an additional constraint on an affine interval, allowing asymmetric intervals to be represented. Further, these bounds also incorporate probabilistic information: the probability of exceeding the asymmetric bounds is less than a user-specified small value. We also develop new algorithms for interval computations that handle the enhanced form of affine intervals. This improvement not only broadens the application of AA, but also improves the accuracy of nonlinear interval computations. We know that many nonlinear computations naturally yield asymmetric intervals, even if the input intervals are center-symmetric. So the ability of representing asymmetric intervals makes nonlinear interval operations more accurate. We have demonstrated that in a series of many nonlinear interval computations, our new interval technique substantially outperforms the original affine arithmetic in terms of accuracy.

- **Introduce a better algorithm for nonlinear binary interval functions**

  In affine arithmetic, nonlinear binary interval functions, such as multiplication and division, used to suffer from low accuracy due to the approximations made during the interval computations. In Chapter 4, we introduced a better approximation algorithm for interval binary functions, called the *minivolume approximation*. It utilizes a geometric interpretation of a binary interval function, and finds the answer in the two bounding planes of a nonlinear surface that is described by the function. This algorithm not only produces a tighter output interval, but also provides a way to accommodate the new asymmetric affine form. This improvement makes interval-valued techniques more promising in applications that are dominated by nonlinear binary functions.

- **Provide means to estimate probability distribution via interval computations**

  Traditionally, interval techniques have been used to provide a bound estimation. However, in many problems, the detailed probability distribution within the bounded interval is more desirable. In Chapter 5, we develop techniques to estimate the probability distribution, which could be center-asymmetric, within an affine interval. It relies on interval computations to identify, in an interval, the highest probable point and the standard deviation to the left and the right to that point, from which an approximate distribution is derived. Unlike other distribution analysis approaches, our approach propagates affine intervals, and hence enjoys the efficient correlation handling of affine arithmetic. Moreover, by heavily utilizing the asymmetric bounds of an interval, we are able to capture the skewness of a distribution. With this technique, interval computations can be adopted in applications where distribution analysis is the main objective. We demonstrate the effectiveness and applicability of our approach in several soft-max applications.

- **Build a C++ library that supports probabilistic interval computations**

  A byproduct of this thesis research is a C++ interval computation library. It overloads the common C++ arithmetic operators with their interval counterparts, which allows us to convert a regular C++ program into an interval one with minimal modification to the source code. Further, the library provides options for choosing the complexity for the interval representation and common interval computations, thus enabling users to explore tradeoffs between accuracy and computational cost.

6.2 Future Work

While our work significantly improves affine arithmetic, there are still many open issues that remain to be addressed in the future. The main limitation with the current techniques is the computational complexity. Since the objective of my thesis is to fully explore the opportunities of accuracy improvement, we have not given much attention to the complexity issue. As we have shown in the soft-max binary tree example in Chapter 5, the interval-valued analysis is only about 20–50X faster than Monte Carlo simulation with 100,000 runs. This speedup is not impressive compared to many other analytical approaches. However, there are at least three directions for further speedup.

One direction for reducing computational cost is to implement heuristic noise symbol manage-

ment. In affine arithmetic, noise symbols are used to retain correlation information. Their presence helps to keep track of correlations and contributes to more accurate interval computations. However, when an application has numerous uncertain quantities or involves many steps of computations, interval analysis can generate a vast number of noise symbols. It becomes a large computational burden to carry them around in interval analysis. Further, the complexities of our interval computation algorithms grow proportionally with the number of noise symbols. Therefore, it is very important to control the number of noise symbols. First, we can eliminate those noise symbols whose corresponding uncertainty terms in all affine intervals have relatively small magnitudes. The corresponding uncertainty terms can be lumped into the most dominant uncertainty term or spread among all other uncertainty terms in an affine interval. Second, for each affine interval, we can identify those uncertainty terms who noise symbols are unique, and "condense" these terms into a single one and replace those old noise symbols with a new one. Third, theoretically speaking, when affine intervals are considered to have normal distributions, N intervals need no more than N independent noise symbols to completely carry their correlation information. These independent noise symbols can be found by performing PCA (Principle Component Analysis). So when the number of noise symbols is much larger than the number of active affine intervals (N) during interval analysis, we can use PCA to find the relationship between the affine intervals and the new set of noise symbols whose size is no larger than N. However, PCA is computationally intensive too. So we need to study when and how often this re-orthogonalization should be conducted in order to remain beneficial.

The second direction for improving computational efficiency is to minimize the chance that new noise symbols are generated during interval computations. In our current algorithm for any nonlinear interval function, a new noise symbols is always generated to account for the approximation error made during the interval operation. Since this new noise symbol is unique to the output interval, we can spread the uncertainty of the approximation error to the other uncertainty terms in the affine interval instead of adding a new noise symbol (a prelimary version of this idea already happens in Ma's work [57, 58]. However, this may alter the correlations between this output and other affine intervals, and eventually hurt the accuracy of subsequent interval computations. We should thoroughly study the gains and the costs of this measure for different interval computations, and identify the situations where avoiding a new noise symbol has negligible impact.

The last, but not least, direction is to optimize the source code of the C++ interval computation library for speed. The current implementation mainly focuses on code structure and readability, and does not give much consideration to code efficiency. We believe significant improvement can be achieved through various code optimization techniques.

The applications we use throughout the dissertation do not have conditional statements, since most DSP applications are dataflow type of algorithms. It remains an interesting topic to study how we should handle conditionals in interval computation. In Ma's work on interconnect modeling [57], conditionals are handled based on central value comparison, which is a reasonable, though imperfect, heuristic. Another possible approach is, to compute the probability of each branch being executed based on the interval version of the conditional statement, and then aggregate the outcomes from all branches using these probabilities.

Another area for future work is to develop a closed-form distribution function. Our current approach presented in Chapter 5 is a heuristic, and it provides a look-up table for a smoothed PDF as the end result. In real applications, it may be more desirable to have a closed-form distribution function. It may be possible to use a more analytical method to construct a "best fitting" distribution function based on the PDF look-up table that we obtain from the asymmetric affine interval.

6.3 Closing Remarks

In conclusion, this dissertation contributes a probabilistic approach to the class of interval methods. It significantly reduces the conservatism of interval computations, especially for nonlinear functions, at the cost of computational complexity. We have no doubt that our contribution will broaden the application of interval methods. However, one should to be aware of the accuracy and complexity tradeoff when choosing interval methods. For example, interval analysis [66] is the least accurate, but the fastest, method; affine analysis [17] is more costly, but outperforms interval analysis in accuracy through efficient correlation handling; and our probabilistic approach is the most complex method, but significantly boosts accuracy for nonlinear functions, and is more flexible than affine analysis in representing center-asymmetric intervals.

We have already been very successful in one recent application: modeling the effects of finite

precision errors in common DSP codes. Our novel represention for ”static” precision analysis of fixed-point and floating-point errors [7, 24] proved to 4–5 orders of magnitude faster than conventional Monte Carlo analysis. Ma has applied these ideas with success in statistical interconnect modeling [57]. We hope our enhanced interval-valued models and the theory underlying their creation will prove useful in a variety of other VLSI and DSP applications.

Appendix A

Proof of Equation (3.4)

For a joint Gaussian distribution, a constant density curve is an ellipse.

$$\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - \frac{2\rho(x-\mu_x)(y-\mu_Y)}{\sigma_x\sigma_y} + \left(\frac{y-\mu_y}{\sigma_y}\right)^2 = K(1-\rho^2) \tag{A.1}$$

The parameter K determines the size of the ellipse. Equation (3.4) states that the probability that a sample is within the ellipse λ equals $1 - e^{-K/2}$.

To prove it, we need to conduct two coordinate transformations (see Figure A.1). The first is to transform the ellipse from coordinate (x, y) to coordinate (u, v) (the axes u and v are along the axes of the ellipse), and the second transformation is from coordinate (u, v) to polar coordinate (r, θ).

A common way to find the axes u and v is through PCA (Principle Component Analysis). Suppose we find the eigenvalues of the covariance matrix $\begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix}$ are σ_u and σ_v, and the corresponding eigenvectors are v_1 and v_2. , then the new coordinates can be converted from the original coordinates using the following relationship:

$$\begin{bmatrix} u \\ v \end{bmatrix} = [v_1, v_2]' \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}. \tag{A.2}$$

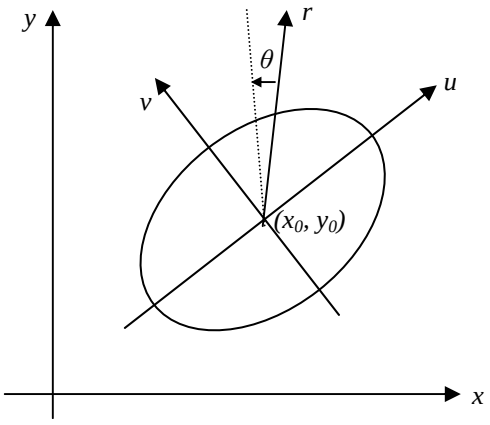


Figure A.1: Coordinate transformation

The eigenvectors v_1 and v_2 are

$$\begin{aligned} v_1 &= \frac{1}{\sqrt{\rho^2\sigma_x^2\sigma_y^2 + (\sigma_u^2 - \sigma_x^2)^2}} \begin{bmatrix} \rho\sigma_x\sigma_y \\ \sigma_u^2 - \sigma_x^2 \end{bmatrix} \\ v_2 &= \frac{1}{\sqrt{\rho^2\sigma_x^2\sigma_y^2 + (\sigma_v^2 - \sigma_x^2)^2}} \begin{bmatrix} \rho\sigma_x\sigma_y \\ \sigma_v^2 - \sigma_x^2 \end{bmatrix} \end{aligned} \tag{A.3}$$

By applying the coordinate transformation, the ellipse under the new coordinate system is written as

$$\frac{u^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2} = K, \tag{A.4}$$

and the joint PDF of u and v is

$$\frac{1}{2\pi\sigma_u\sigma_v} e^{-(\frac{u^2}{2\sigma_u^2} + \frac{v^2}{2\sigma_v^2})}. \tag{A.5}$$

The integration of the joint PDF inside the ellipse is the probability λ we seek.

$$\lambda = \iint_{u,v \in ellipse} \frac{1}{2\pi\sigma_u\sigma_v} e^{-(\frac{u^2}{2\sigma_u^2} + \frac{v^2}{2\sigma_v^2})} du dv. \tag{A.6}$$

Let $u = r\sigma_u \cos\theta$ and $v = r\sigma_v \sin\theta$. Then

$$\begin{aligned} \lambda &= \int_0^{2\pi} \int_0^{\sqrt{K}} \frac{1}{2\pi\sigma_u\sigma_v} e^{-\frac{r^2}{2}} \sigma_u\sigma_v r dr d\theta \\ &= \int_0^{\sqrt{K}} e^{-\frac{r^2}{2}} r dr \\ &= 1 - e^{-\frac{K}{2}} \end{aligned} \tag{A.7}$$

which is what we sought to show.

Appendix B

Solution to Equation (4.17)

Equation (4.17) is

$$\frac{d(uv)}{dv} = 0. \tag{B.1}$$

where u and v have the following relationship

$$au^2 + bv^2 + cuv = 1 \; a, b \neq 0, \tag{B.2}$$

or

$$u = -\frac{c}{2a}v \pm \frac{1}{2a}\sqrt{(c^2 - 4ab)v^2 + 4a}. \tag{B.3}$$

By using (B.3), equation (4.17) can be rewritten as

$$\frac{d(-\frac{c}{2a}v^2 \pm \frac{v}{2a}\sqrt{(c^2 - 4ab)v^2 + 4a})}{dv} = 0, \tag{B.4}$$

In this appendix, we provide the values of u and v that satisfy this equation.

Equation (**??**) can be first turned into

$$-\frac{c}{a}v \pm \frac{(c^2 - 4ab)v^2 + 2a}{a\sqrt{(c^2 - 4ab)v^2 + 4a}} = 0. \tag{B.5}$$

Then, the above equation can be further converted to the following

$$v^4 + \frac{4a}{c^2 - 4ab}v^2 - \frac{a}{bc^2 - 4ab^2} = 0, \tag{B.6}$$

or

$$v^4 + Av^2 + B = 0, \tag{B.7}$$

where $A = \frac{4a}{c^2-4ab}$ and $B = -\frac{a}{bc^2-4ab^2}$. This is a special 4th-degree equation, with four easy-to-compute roots.

Keep in mind that the constants a, b, c come from the parameters of a joint Gaussian distribution. More specifically,

$$\begin{aligned} a &= \frac{1}{K(1-\rho^2)\sigma_1^2} \\ b &= \frac{1}{K(1-\rho^2)\sigma_2^2} \\ c &= -\frac{2\rho}{K(1-\rho^2)\sigma_1\sigma_2}. \end{aligned} \tag{B.8}$$

By using these constraints, we can easily see that the four roots of (B.7) are all real. They are

$$\begin{aligned} v_{1,2} &= \pm\sqrt{-\frac{A}{2} - \frac{1}{2}\sqrt{A^2-4B}} = \pm\sqrt{-\frac{2a-c\sqrt{a/b}}{4ab-c^2}} \\ v_{3,4} &= \pm\sqrt{-\frac{A}{2} + \frac{1}{2}\sqrt{A^2-4B}} = \pm\sqrt{-\frac{2a+c\sqrt{a/b}}{4ab-c^2}} \end{aligned} \tag{B.9}$$

With the four roots in (B.9) and the relationship in (B.3), we can find the four solution points to equation (4.17). They are

$$\begin{aligned} u_1 &= \frac{1}{2a}(1 - c/\sqrt{4ab-c^2})\sqrt{2a-c\sqrt{a/b}} \\ v_1 &= \sqrt{(2a-c\sqrt{a/b})/(4ab-c^2)} \\ u_2 &= \frac{1}{2a}(-1 - c/\sqrt{4ab-c^2})\sqrt{2a-c\sqrt{a/b}} \\ v_2 &= -\sqrt{(2a-c\sqrt{a/b})/(4ab-c^2)} \\ u_3 &= \frac{1}{2a}(1 - c/\sqrt{4ab-c^2})\sqrt{2a+c\sqrt{a/b}} \\ v_3 &= \sqrt{(2a+c\sqrt{a/b})/(4ab-c^2)} \\ u_4 &= \frac{1}{2a}(-1 - c/\sqrt{4ab-c^2})\sqrt{2a+c\sqrt{a/b}} \\ v_4 &= -\sqrt{(2a+c\sqrt{a/b})/(4ab-c^2)}. \end{aligned} \tag{B.10}$$

Appendix C

Parameters in Equation (4.21)

Equation (4.21) asks us to solve

$$\frac{d(xy)}{dy} = 0,$$

where

$$xy = x_0 y - \frac{c}{2a} y(y - y_0) \pm \frac{y}{2a}\sqrt{(c^2 - 4ab)(y - y_0)^2 + 4a}. \tag{C.1}$$

In this appendix, we convert (4.21) into a 4th-degree equation and provide the detailed parameters.

Let $d = c^2 - 4ab$. Then

$$\frac{d(xy)}{dy} = x_0 + \frac{c}{2a} y_0 - \frac{c}{a} y \pm \left(\frac{1}{2a}\sqrt{d(y - y_0)^2 + 4a} + \frac{y}{2a}\frac{d(y - y_0)}{\sqrt{d(y - y_0)^2 + 4a}}\right) = 0 \tag{C.2}$$

The equation is equivalent to

$$-x_0 - \frac{c}{2a} y_0 + \frac{c}{a} y = \pm\left(\frac{1}{2a}\sqrt{d(y - y_0)^2 + 4a} + \frac{y}{2a}\frac{d(y - y_0)}{\sqrt{d(y - y_0)^2 + 4a}}\right) = 0 \tag{C.3}$$

Then we multiply the both sides with $2a\sqrt{d(y - y_0)^2 + 4a}$, and get

$$(-2ax_0 - cy_0 + 2cy)\sqrt{d(y - y_0)^2 + 4a} = \pm(d(y - y_0)^2 + 4a + dy(y - y_0)) \tag{C.4}$$

Let $e = 2ax_0 + cy_0$ and $f = 4a + dy_0^2$. The above equation can be further converted into the following 4th-degree equation:

$$Ay^4 + By^3 + Cy^2 + Dy + E = 0, \tag{C.5}$$

where

$$\begin{aligned} A &= 4c^2d - 4d \\ B &= -8c^2dy_0 - 4cde + 12d^2y_0 \\ C &= 4c^2f + de^2 + 8cdey_0 - 4df - 9d^2y_0^2 \\ D &= -2de^2y_0 - 4cef + 6dfy_0 \\ E &= e^2f - f^2. \end{aligned} \tag{C.6}$$

The analytical solution to this 4th-degree equation can be found in [96]. We give the results in the following. Let

$$\begin{aligned} m &= c^2 - 3bd + 12ae \\ n &= 2c^3 - 9bcd + 27ad^2 + 27b^2e - 72ace \\ o &= -4m^3 + n^2 \\ p &= (n + \sqrt{o})^{1/3} \\ q &= -(b^3)/(a^3) + 4bc/(a^2) - 8d/a \\ r &= b^2/(4a^3) - 2c/(3a) \\ s &= 2^{1/3}m/(3ap) + p/(3 \cdot 2^{1/3}a) \\ t &= \sqrt{r+s} \\ u &= 2r - s \\ v &= -b/(4a) \\ w1 &= \sqrt{u - q/(4t)} \\ w2 &= \sqrt{u + q/(4t)}, \end{aligned} \tag{C.7}$$

and the four roots are

$$\begin{aligned} y_1 &= v - 0.5t - 0.5w1 \\ y_2 &= v - 0.5t + 0.5w1 \\ y_3 &= v + 0.5t - 0.5w2 \\ y_4 &= v + 0.5t + 0.5w2 \end{aligned} \tag{C.8}$$

Note that some of the roots of this equation may be complex. In our problem, we are interested in only the real roots.

Appendix D

Distribution of the ratio between two correlated normal random variables

A special case of this problem, i.e. the distribution of the ratio between two *independent standard normal* random variables, has been covered in [61]. In this appendix, we solve the general case, i.e. the distribution of the ratio between two normal correlated random variables. It is achieved by developing a transformation between these two cases.

First, we briefly summarize the result in [61]. Suppose

$$z' = \frac{a + x'}{b + y'} \tag{D.1}$$

where a and b are non-negative constants, and x' and y' are independent standard normal random variables. If b is large, say $b > 3$ (this condition means that the denominator in (**??**) is very unlikely to be zero), then the CDF of z' can be computed from the standard normal CDF:

$$P(z' < t) = \int_{-\infty}^{\frac{bt-a}{\sqrt{1+t^2}}} \phi(x)dx = \Phi(\frac{bt - a}{\sqrt{1 + t^2}}), \tag{D.2}$$

where $\phi(x)$ is the standard normal PDF and $\Phi(x)$ is the standard normal CDF.

Now, let us look at the more general case, which is

$$z = \frac{x}{y},$$

where x and y are *correlated* normal random variables. If we could find an invertible function f such that $z = f(z')$ (z' complies with the form in (D.5)), then we can compute the distribution of z from the distribution of w. We obtain the function f using the Cholesky decomposition.

Suppose $x \sim \mathcal{N}(x_0, \sigma_x^2)$, $y \sim \mathcal{N}(y_0, \sigma_y^2)$, and the correlation coefficient between x and y is ρ. Then by performing the Cholesky decomposition on the covariance matrix, we obtain the following transformation

$$\begin{cases} x = & x_0 + \rho\sigma_x y' + \sqrt{1-\rho^2}\sigma_x x' \\ y = & y_0 + \sigma_y y', \end{cases}$$

where x' and y' are independent standard normal random variables. As a result, the ratio z can be rewritten as

$$\begin{aligned} z &= \frac{x_0 + \rho\sigma_x y' + \sqrt{1-\rho^2}\sigma_x x'}{y_0 + \sigma_y y'} \\ &= \rho\frac{\sigma_x}{\sigma_y} + \sqrt{1-\rho^2}\frac{\sigma_x}{\sigma_y}\frac{\frac{x_0 - y_0\rho\sigma_x/\sigma_y}{\sqrt{1-\rho^2}\sigma_x} + x'}{y_0/\sigma_y + y'} \\ &= c_1 + c_2\frac{a + x'}{b + y'}, \end{aligned} \tag{D.3}$$

where a, b, c_1, c_2 are

$$\begin{aligned} a &= \left|\frac{x_0 - y_0\rho\sigma_x/\sigma_y}{\sqrt{1-\rho^2}\sigma_x}\right| \\ b &= \left|\frac{y_0}{\sigma_y}\right| \\ c_1 &= \rho\frac{\sigma_x}{\sigma_y} \\ c_2 &= \begin{cases} \sqrt{1-\rho^2}\frac{\sigma_x}{\sigma_y} & \text{when } (x_0 - y_0\rho\sigma_x/\sigma_y)y_0 \geq 0 \\ -\sqrt{1-\rho^2}\frac{\sigma_x}{\sigma_y} & \text{otherwise} \end{cases} \end{aligned} \tag{D.4}$$

Now, let $z' = \frac{a+x'}{b+y'}$ which follows the form in (D.5), and we obtain a linear transformation between the general case z and the special case z', i.e.,

$$z = c_1 + c_2 z' \tag{D.5}$$

Finally, based on (D.2) and (D.5), we can compute the CDF of z as the following. When $c_2 > 0$

$$\begin{aligned} P(z < s) &= P(c_1 + c_2 z' < s) \\ &= P(z' < \frac{s - c_1}{c_2}) \\ &= \Phi(\frac{b(\frac{s-c_1}{c_2}) - a}{\sqrt{1 + (\frac{s-c_1}{c_2})^2}}), \end{aligned} \tag{D.6}$$

otherwise,

$$\begin{aligned} P(z < s) &= P(c_1 + c_2 z' < s) \\ &= P(z' > \frac{s - c_1}{c_2}) \\ &= 1 - P(z' < \frac{s - c_1}{c_2}) \\ &= 1 - \Phi(\frac{b(\frac{s-c_1}{c_2}) - a}{\sqrt{1 + (\frac{s-c_1}{c_2})^2}}) \end{aligned} \tag{D.7}$$

Appendix E

Modeling Gate Delay with Soft-Max

In circuit timing analysis, logic gates are usually modeled as max functions of the inputs' arrival times. However, the max-delay model tends to underestimate the output's arrival time when the inputs are switching simultaneously or in temporal proximity. In this appendix, we provide experimental evidences on the temporal proximity effects and propose a more accurate soft-max-delay model.

E.1 Motivation: Underestimation by the max-delay model

For a two-input logic gate, the conventional max-delay model describes the output arrival time as

$$t_o = \max(t_1 + d_1, t_2 + d_2),$$

where d_1 and d_2 are pin-to-pin delays, and t_1 and t_2 are the two inputs' arrival times. However, when the two inputs switch simultaneously or in temporal proximity, this model fails to accurately estimate the output arrival time. Here we use a NAND gate to show the temporal proximity effects. Let the separation time between the two inputs to be $t_{sep} = t_1 - t_2$. In Figure E.1, we plot how the output arrival time changes with respect to t_{sep} and compare it with the max-delay model. We can see that unlike the max function, the output arrival time is a *smooth* function of the separation time. When $|t_{sep}|$ is large, the max-delay model provides an accurate estimate for the output arrival time. However, this model has significant underestimation in the proximity of $|t_{sep}| = 0$. We characterize

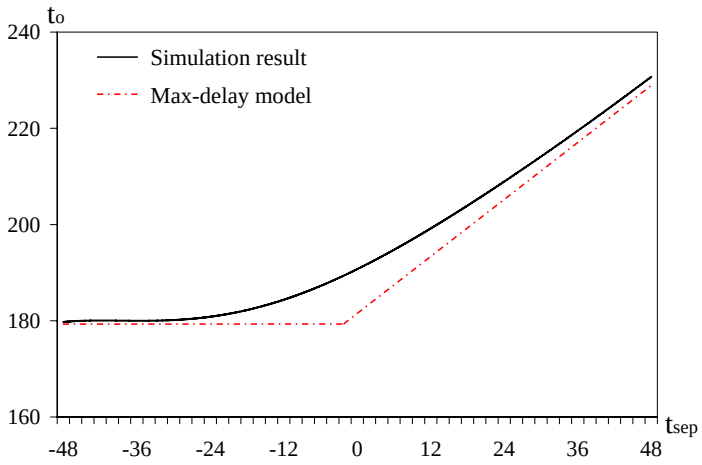


Figure E.1: Simulation result compared to the max-delay model (NAND gate)

The x-axis is the separation between the two inputs' arrival times, and the y-axis is the output arrival time.

the underestimation by the maximum discrepancy between the modeled arrival time and the actual arrival time, normalized by the average pin-to-pin delay:

$$\text{error} = \frac{\max\{t_o^{actual} - t_o^{modeled}\}}{(d_1 + d_2)/2}.$$

In this experiment, the output load is $0.01pF$ and the input slope is $60ns$, we find that the underestimation reaches about 20%.

Further, the underestimation becomes more severe as the input slope increases. Table E.1 shows how the underestimation is affected by the slope change. We fix the output load $C_l = 0.02pF$, and vary the input slope from $30ns$ to $90ns$. Accordingly, the underestimation is increased from 13.8% to 26%.

slope (ns)	30	60	90
error	13.8%	19.2%	26.0%

Table E.1: Underestimation vs. input slope ($C_l = 0.02pF$)

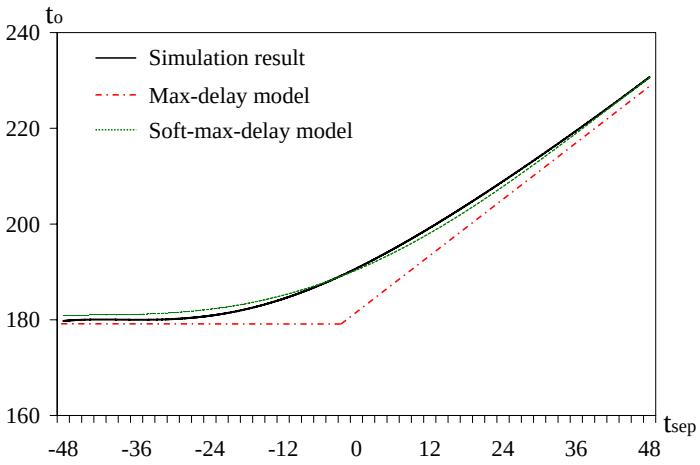


Figure E.2: Soft-max modeling results for a NAND gate

E.2 Soft-max modeling for NAND gates

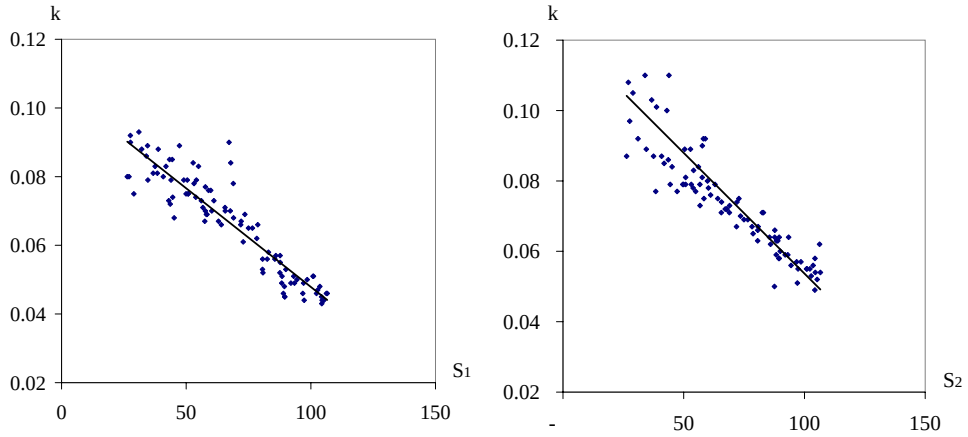
We conduct experiments on a 2-input CMOS NAND gate. The sizes of the NMOS and the PMOS transistors are chosen to ensure the equal rise and the fall delays. We simulate the circuit with the 180nm technology models, and measure the input and output arrival times using $0.8V_{dd}$ (for rising edge) and $0.2V_{dd}$ (for falling edge) as the thresholds. To describe the relationship between the output arrival time t_o and the input arrival times t_1 and t_2, we propose the following soft-max-delay model:

$$\begin{aligned} t_o &= \text{smax}(t_1 + d_1, t_2 + d_2) \\ &= \frac{1}{k}\log(e^{k(t_1+d_1)} + e^{k(t_2+d_2)}). \end{aligned} \tag{E.1}$$

The difference between this new model and the conventional max-delay model is significant only when $t_1 + d_1$ and $t_2 + d_2$ are close.

Note that in (E.1), the parameter k controls the "sharpness" of this function: the larger the k, the closer it is to the max function. We determine the value of k by fitting the experimental data with this soft-max function. The best results are shown in Figure E.2, where we compare the output arrival time estimated by the new model with the simulation result estimated by the max-delay model. We see that soft-max model closely matches the simulation result, with less than 5% error.

Interestingly, the parameter k that best fits the simulation result varies with the slopes of the input signals, and there appears to be a linear dependency between them. We define slope to be the transition time between $0.2V_{dd}$ and $0.8V_{dd}$. Suppose the two input slopes of a NAND gate are s_1



(a) k vs. s_1 (b) k vs. s_2

Figure E.3: The model parameter k vs. input slopes

and s_2. We first fix s_2, and study how k changes with s_1. The results are shown in Figure E.3(a). We see that as the input slope s_1 gets larger, the best k becomes smaller, or in other words, the soft-max function that best describes the output arrival time becomes "softer". The data in the figure clearly shows an inverse linear relationship between them. Similarly, by fixing s_1 and varying s_2, we find k and s_2 also also has an inverse linear dependency.

Ideally, we would like to build a function for k to describe its dependency on all related physical parameters. However, due to the scope of this thesis, we do not pursue this direction further. We believe, with sufficient amount of simulation data, developing a model for k is not a very difficult task.

E.3 Soft-max modeling for NOR gates

Similar to NAND gates, the max-delay model for NOR gates is also inaccurate in case of multiple signal switching. However, the relationship between the output arrival time and the inputs' arrival times is more complicated than that of a NAND gate. Our experiments are on a 2-input CMOS NOR gate, with equal rising and falling delays. We denote the discrepancy between the two inputs' arrival times by $t_{sep} = t_1 - t_2$, and plot the output arrival times for various t_{sep}'s in Figure E.4. When $|t_{sep}|$ is large, the output arrival time is exactly what is estimated by the max-delay model.

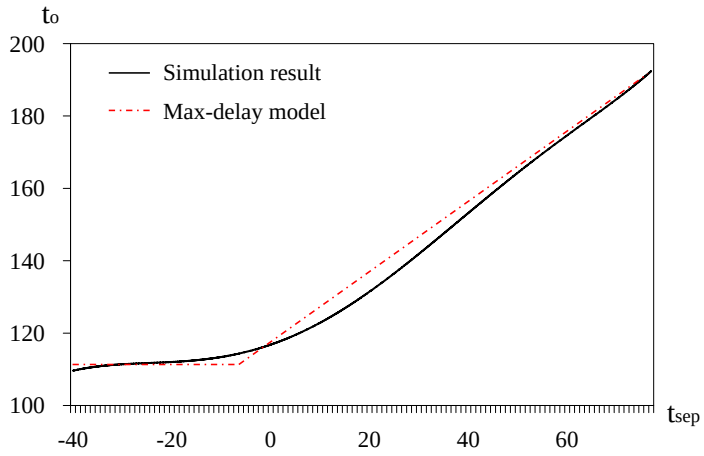


Figure E.4: Simulation result compared to the max-delay model (NOR gate)

The x-axis is the separation between the two inputs' arrival times, and the y-axis is the output arrival time.

However, when $|t_{sep}|$ is in the proximity of zero, the output arrival time is neither a max function of the inputs' arrival times, nor a soft max function as we have seen in NAND gates. We observe two characteristics about the curve: first, it is similar to a soft-max curve that is shifted towards the right, and second, the 2nd derivative of this curve crosses zero at some point since it starts from being very close to the max function, then it goes above it and then under it, and eventually, approaches the max function again.

Based on the curve shape, we propose the following model which is a combination of a soft-max function and a sigmoid function:

$$t_o = \frac{1}{k}\log(\exp^{k(t_1+d_1-a)} + \exp^{k(t_2+d_2)}) + \frac{b}{1+e^{-c((t_{sep}-d)}}. \tag{E.2}$$

This function approaches the max-delay model when $|t_{sep}|$ is large. We determine the parameters a, b, c, d and k by fitting the experimental data with this function. The best fit is shown in Figure E.5. We see that the new function in (E.2) matches the simulation result very well, with less than 3% error.

Note that the model in (E.2) has five parameters, all of which may vary with some physical parameters of a logic gate, such as slopes, capacitive load, gate width and length, ..., etc. This raises an intriguing question: can we effectively find models for these parameters, and does the benefit of using the more complicated soft-max model justify the extra cost of searching for the best

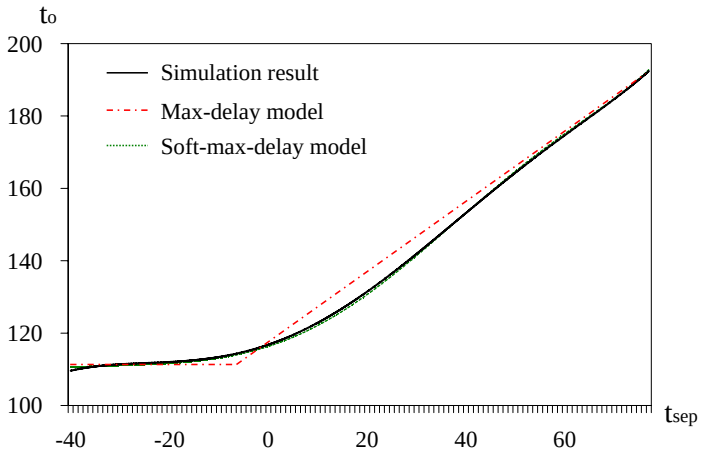


Figure E.5: Soft-max modeling results for a NOR gate

parameters. More thorough study on gate delays is needed to answer this question.

E.3.1 Summary

In this appendix, we examine the relationship between a logic gate's output arrival time and the inputs' arrival time. Through experiments, we demonstrate that the conventional max-delay model for NAND gates and NOR gates is inaccurate in case of multiple signal switching. Further, we propose more accurate models that utilize the soft-max function to estimate the output arrival time for NAND gates and NOR gates. These models have the following common form:

$$t_o = \frac{1}{k}\log(\exp^{k(t_1+d_1-a)} + \exp^{k(t_2+d_2)}) + \frac{b}{1+e^{-c((t_{sep}-d)}},$$

where a, b, c, d and k are model parameters. In particular, for NAND gates, the parameter b equals to zero. Our experimental data shows that the new model fits the simulation results very well, for both multiple signal switching and single signal switching. However, how to obtain the best parameters still remains an open issue. It will rely on data mining techniques to find the dependency between these model parameters and the physical parameters of a logic gate.

Bibliography

[1] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *Proc. Int. Conf. Computer Aided Design*, 2003.

[2] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda. Statistical delay computation considering spatial correlations. In *Proc. Asia South Pacific Design Automation Conference*, 2003.

[3] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic press, 1983.

[4] A. Benedetti and P. Perona. Bit-width optimization for configurable DSP's by multi-interval analysis. In *Proc. of the 34th Asilomar Conference on Signals, Systems, and Computers*, 2000.

[5] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2003.

[6] R. Burch, F. Najm, P. Yang, and T. Trick. McPOWER: A Monte Carlo approach to power estimation. In *Proc. Int. Conf. Computer Aided Design*, 1992.

[7] C. F. Fang and R. A. Rutenbar and M. Püschel and T. Chen. Towards efficient static analyis of finite precision effects in DSP applications via affine arithmetic modeling. In *Proc. Design Automation Conference*, 2003.

[8] C-XSC—A C++ Class Library for Extended Scientific Computing, http://www.rz.uni-karlsruhe.de/ iam/html/language/cxsc/cxsc.html.

[9] D. Cachera and T. Risset. Advances in bit width selection methodology. In *Proc. Int. Conf. Application-Specific Systems, Architectures, and Processors*, 2002.

[10] Cadence. *FPGA design with Cadence SPW*, 2002.

[11] C. Carreras, J. A. López, and O. Nieto-Taladriz. Bit-width selection for data-path implementations. In *Proc. Int. Symp. System Synthesis*, 1999.

[12] V. Chandramouli and K. A. Sakallah. Modeling the effects of temporal proximity of input transitions on gate propagation delay and transition time. In *Proc. Design Automation Conference*, 1996.

[13] H. Chang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using single pert-like traversal. In *Proc. Int. Conf. Computer Aided Design*, 2003.

[14] M. L. Chang and S. Hauck. Précis: A design-time precision analysis tool. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 2002.

[15] C. E. Clark. The greatest of a finite set of random variables. *Operations Research*, 2, 1961.

[16] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A methodology and design environment for DSP asic fixed-point refinement. In *Proc. Design, Automation and Test in Europe Conf.*, 1999.

[17] J. L. D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. In *Proc. VI Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'93)*, October 1993.

[18] G. A. Constantinides. Perturbation analysis for word-length optimization. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, April 2003.

[19] G. A. Constantinides, P. Y.K. Cheung, and W. Luk. The multiplewordlength paradigm. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, April 2001.

[20] L. H. de Figueiredo. Surface intersection using affine arithmetic. In *Proceedings of Graphics Interface '96*, May 1996.

[21] L. H. de Figueiredo, R. Van Iwaarden, and J. Stolfi. Fast interval branch-and-bound methods for unconstrained global optimization with affine arithmetic. Technical Report IC-97-08, Institute of Computing, Univ. of Campinas, June 1997.

[22] L. H. de Figueiredo and J. Stolfi. Self-validated numerical methods and applications. *Brazilian Mathematics Colloquium monograph, IMPA, Rio de Janeiro, Brazil*, July 1997.

[23] A. Devgan and C. Kashyap. Block-based static timing anlaysis with uncertainty. In *Proc. Int. Conf. Computer Aided Design*, 2003.

[24] C. F. Fang, R. A. Rutenbar, and T. Chen. Fast, accurate static analyis for fixed-point finite-precision effects in DSP designs. In *Proc. Int. Conf. Computer Aided Design*, 2003.

[25] F. Fang, T. Chen, and R. A. Rutenbar. Floating-point bit-width optimization for low-power signal processing applications. In *Proc. Int. Conf. Acoustic, Speech, and Signal Processing*, May 2002.

[26] F. Fang, T. Chen, and R. A. Rutenbar. Lightweight floating-point arithmetic: Case study of inverse discrete cosine transform. *EURASIP J. Sig. Proc.: Special Issue on Applied Implementation of DSP and Communication Systems*, 2002(9):879–892, September 2002.

[27] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley, New York, 3 edition, 1971.

[28] N. Femia and G. Spagnuolo. True worst-case circuit tolerance analysis using genetic algorithm and affine arithmetic. *IEEE Trans. on Circuits and Systems*, 47(9):1285–1296, September 2000.

[29] A. A. Gaffar, O. Mencer, W. Luk, and P. Y.K. Cheung. Unifying bit-width optimisation for fixed-point and floating-point designs. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, April 2004.

[30] A. A. Gaffar, O. Mencer, W. Luk, P. Y.K. Cheung, and N. Shirazi. Floating-point bitwidth analysis via automatic differentiation. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, April 2002.

[31] S. Goldenstein, C. Vogler, and D. Metaxas. Affine arithmetic based estimation of cue distributions in deformable model tracking. In *Proceedings of 2001 Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, December 2001.

[32] S. Goldenstein, C. Vogler, and D. Metaxas. Cue integration using affine arithmetic and gaussians. Technical Report MS-CIS-02-06, University of Pennsylvania, 2002.

[33] Mentor Graphics. *DSP Station C code generation user's manual*, 1996.

[34] E. Hansen. A generalized interval arithmetic. *Interval Mathematics, Lecture Notes in Computer Science*, (29):7–18, 1975.

[35] E. R. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, Inc., 1992.

[36] C. L. Harkness. *An Approach to Uncertainty in VLSI Design*. PhD thesis, Brown University, May 1991.

[37] C. L. Harkness and D. P. Lopresti. Interval methods for modeling uncertainty in RC timing analysis. *IEEE Trans. Computer-Aided Design*, 11(11), November 1992.

[38] E. Henson. *Topics in Interval Analysis*. Oxford Unversity Press, 1969.

[39] INTLAB—INTerval LABoratory, http://www.ti3.tu-harburg.de/rump/intlab.

[40] J. E. Jackson. *A User's Guide to Principle Components*. New York: Wiley-Interscience, 1991.

[41] Y. H. Jun, K. Jun, and S. B. Park. An accurate and efficient delay time modeling for MOS logic circuits using polynomial approximation. *IEEE Trans. Computer-Aided Design*, 9(6):1027–1032, 1989.

[42] S. M. Kang. Accurate simulation of power dissipation in VLSI circuits. *IEEE Journal of Solid-State Circuits*, SC-21(5):889–891, 1986.

[43] H. Keding, M. Willems, M. Coors, and H. Meyr. FRIDGE: a fixed-point design and simulation environment. In *Proc. Design, Automation and Test in Europe Conf.*, 1998.

[44] S. Kim and E. A. Lee. Infrastructure for numeric precision control in the Ptolemy environment. In *Proc. of the 40th Midwest Symposium on Circuits and Systems*, August 1997.

[45] S. Kim and W. Sung. Fixed-point error analysis and word length optimization of 8X8 idct architecture. *IEEE Trans. Circuits and Systems for Video Technology*, 8(8), December 1998.

[46] F. Korn and Ch. Ullrich. Verified solution of linear systems based on common software libraries. *Interval Computations*, 3:116–132, 1993.

[47] W. Kramer. A priori worst case error bounds for floating-point computations. *IEEE Trans. Computer*, 47:750–756, July 1998.

[48] K. I. Kum and W. Sung. AUTOSCALER for C: an optimization floating-point to integer C program converter for fixed-point digital signal processing. *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, 47(9), September 2000.

[49] K. I. Kum and W. Sung. Combined word-length optimization and high-level synthesis of digital signal processing systems. *IEEE Trans. Computer-Aided Design*, 20(8), August 2001.

[50] J. Le, X. Lin, and L. T. Pileggi. STAC: Statistical timing analysis with correlation. In *Proc. Design Automation Conference*, 2004.

[51] A. Lemke, L. Hedrich, and E. Barke. Analog circuit sizing based on formal methods using affine arithmetic. In *Proc. Int. Conf. Computer Aided Design*, pages 486–489, November 2002.

[52] M. P. Leong, M. Y. Yueng, C. K. Yueng, C. W. Fu, P. A. Heng, and P. H. W. Leong. Automatic floating to fixed point translation and its application to post-rendering 3D warping. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, April 1999.

[53] X. Li, J. Le, P. Gopalakrishnan, and L. Pileggi. Asymptotic probability extraction for non-normal distributions of circuit performance. In *Proc. Int. Conf. Computer Aided Design*, November 2004.

[54] J. Liou, K. Cheng, S. Kundu, and A. Krstic. Fast statistical timing analysis by probabilistic event propagation. In *Proc. Design Automation Conference*, 2001.

[55] B. Liu and T. Kaneko. Error analysis of digital filters realized with floating-point arithmetic. *Proc. IEEE*, 57:1735–1747, October 1969.

[56] Y. Liu, L. Pileggi, and A. J. Strojwas. Model order-reduction of RC(L) interconnect including variational analysis. In *Proc. Design Automation Conference*, June 1999.

[57] J. D. Ma and R. A. Rutenbar. Interval-valued reduced order statistical interconnect modeling. In *Proc. Int. Conf. Computer Aided Design*, November 2004.

[58] J. D. Ma and R. A. Rutenbar. Fast intervalvalued statistical interconnect modeling and reduction. In *International Symposium on Physical Design*, April 2005.

[59] Y. Ma. An accurate error analysis model for fast fourier transform. *IEEE Trans.Signal Processing*, 45(6), June 1997.

[60] S. Mahlke, R. Ravindran, M. Schlanser, R. Schreiber, and T. Sherwood. Bitwidth cognizant architecture synthesis of custom hardware accelerators. *IEEE Trans. Computer-Aided Design*, 20(11), November 2001.

[61] G. Marsaglia. Ratios of normal variables and ratios of sums of uniform variables. *Journal of the American Statistical Association*, 60(309):193–204, March 1965.

[62] Mathworks. *Fixed-point blockset user's guide (ver. 2.0)*, 1999.

[63] W. Q. Meeker and L. A. Escobar. An algorithm to compute the cdf of the product of two normal random variables. *Communications in Statistics*, 23:271–280, 1994.

[64] D. Menard and O. Sentieys. Automatic evaluation of the accuracy of fixed-point algorithms. In *Proc. Design, Automation and Test in Europe Conf.*, 2002.

[65] F. Messine and A. Mahfoudi. Use of affine arithmetic in interval optimization algorithms to solve multidimensional scaling problems. In *Proceedings of SCAN'98–IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, pages 22–25, September 1998.

[66] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[67] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, 1979.

[68] A. Nabavi-Lishi and N. C. Rumin. Inverter models of CMOS gates for supply current and delay evaluation. *IEEE Trans. Computer-Aided Design*, 13(10):1271–1279, 1994.

[69] S. R. Nassif. Modeling and forecasting of manufacturing variations. In *2000 5th International Workshop on Statistical Methodology*, June 2000.

[70] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[71] A. V. Oppenheim and C. J. Weinstein. Effects of finite register length in digital filtering and the fast fourier transform. *Proc. IEEE*, 60(8):957–976, 1972.

[72] M. Pauwels, D. Lanneer, F. Catthoor, G. Goossens, and H. De Man. Models for bit-true simulation and high-level synthesis of DSP applications. In *IEEE Great Lakes Symp. on VLSI*, February 1992.

[73] A. B. Poritz. Hidden markov models: a guided tour. In *Proc. Int. Conf. Acoustic, Speech, and Signal Processing*, 1988.

[74] PROFIL/BIAS—Programmer's Runtime Optimized Fast Interval Library, http://www.ti3.tu-harburg.de/software/profilenglisch.html.

[75] M. Püschel, B. Singer, J. Xiong, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, and R. W. Johnson. SPIRAL: a generator for platform-adapted libraries of signal processing algorithms. *to appear in Journal of High Performance Computing and Applications*.

[76] B. D. Rao. Floating point arithmetic and digital filters. *IEEE Trans.Signal Processing*, 40:85–95, January 1992.

[77] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Horwood, 1984.

[78] H. Ratschek and J. Rokne. *New Computer Mehods for Global Optimization*. Wiley, 1988.

[79] H. Schwandt. An interval arithmetic approach for an almost globally convergent method for the solution of the nonlinear Poisson equation. *SIAM Journal on Scientific and Statistical Computing*, 5(2):427–452, 1984.

[80] H. Schwandt. An interval arithmetic method for the solution of nonlinear systems of equations on a vector computer. *Parallel Computing*, 4(3):323–337, 1987.

[81] H. H. Shou, R. Martin, I. Voiculescu, and G. Wang. Affine arithmetic in matrix form for polynomial evaluation and algebraic curve drawing. *Progress in Natural Science*, 12(1):77–81, January 2002.

[82] M. R. Spiegel. *Theory and Problems of Probability and Statistics*. McGraw-Hill, New York, 1992.

[83] G. W. Steward. The decompositional approach to matrix computation. *Computing in Science and Engineering*, 2:50–59, January–Feburary 2000.

[84] Synopsis. *Converting ANSI-C into fixed-point using Cocentric Fixed-Point Designer*, April 2000.

[85] http://www.systemc.org.

[86] J. Y.F. Tong, D. Nagle, and R. A. Rutenbar. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *EEE Trans. VLSI Syst.*, 8(3), June 2000.

[87] C. Visweswariah, K. Ravindran, and K. Kalafala. First-order parameterized block-based statistical timing analysis. In *Proc. Int. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems(TAU)*, 2004.

[88] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Proc. Design Automation Conference*, 2004.

[89] J. Vygen. Algorithms for detailed placement of standard cells. In *Proc. Design, Automation and Test in Europe Conf.*, 1998.

[90] S. A. Wadekar and Alice C. Parker. Accuracy sensitive word-length selection for algorithm optimization. In *Proc. Int. Conf. Computer Design*, 1998.

[91] M. P. Wand and M. C. Jones. *Kernel Smoothing Monographs on Statistics and Applied Probability*. Chapman and Hall, 1995.

[92] C. Weinstein and A. V. Oppenheim. A comparison of roundoff noise in floating point and fixed point digital filter realizations. *Proc. IEEE*, 57:1181–1183, June 1969.

[93] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall, 1963.

[94] G. Y. Yacoub and W. H. Ku. An accurate simulation technique for short-circuit power dissipation based on current component isolation. In *IEEE International Symposium on Circuits and Systems*, 1989.

[95] I. D. Yun and S. U. Lee. On the fixed-point-error analysis of several fast DCT algorithms. *IEEE Trans. Circuits and Systems for Video Technology*, 3(1), February 1993.

[96] D. Zwillinger. *CRC Standard Mathematical Tables and Formulae, 30th Edition*. Chemical Rubber Company Press, 1995.