Table-based Device Modeling: Methods and Applications

Archit Gupta

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-66
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-66.html

May 11, 2018

Table-based Device Modeling: Methods and Applications

Archit Gupta

Project Report

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination.

Committee:

Professor Jaijeet S. Roychowdhury
Research Advisor

May 11, 2018

*****

Professor Murat Arcak
Second Reader

May 11, 2018

Abstract

When simulating a practical present day analog circuit, device model evaluations alone can take several days of compute time. This is largely because of the complexity of the physical devices that these models represent. These models typically have only a few inputs, so approximating them with polynomials is feasible and attractive. In this thesis, we build a general purpose framework for translating any given compact model into a table-based approximation. We show that with different interpolants, various improvements can be achieved over conventional analytically-derived 'compact models'. Low order cubic splines provide multiple orders of magnitude in speed improvement. Chebyshev polynomials implemented using Barycentric-Lagrange Interpolation can yield near machine-precision in terms of accuracy while still yielding a significant speedup. Chebyshev polynomials can also be used to diagnose hard-to-find modeling errors, like derivative discontinuities. We also discuss the construction of table-based models from sparse measurements of compact models using compressed sensing.

In order to make this work accessible, **all the code that was used for this report has been released along with it in the form of a MATLAB-based software package titled STEAM.** Examples have been included in the package to reproduce the main results and figures presented here.

Acknowledgement

I am grateful to Prof. Jaijeet S. Roychowdhury for his guidance throughout the work that went into the thesis. Tianshi Wang, and Ahmet Gokcen Mahmutoglu gave valuable ideas and inputs without which this would not have been possible. I would also like to thank the NEEDS project which introduced me to device modeling and is one of the main reasons why this project started off; SRC, whose members gave valuable feedback from time to time and encouraged this work. The Xyce team at Sandia National labs was also very kind in providing comments and feedback on various aspects of device modeling, compressive sensing and circuit simulation at large. Finally, I thank my family and friends who have always been very supportive. Without them, this would have never happened.

Contents

1 Introduction and Overview

Simulation of analog components in a system is computationally intense, typically requiring days or even months for a single run over a small set of trial inputs. For example, simulating an Analog-to-Digital Converter (ADC) comprising roughly 50,000 transistors (each represented with a Penn-State Phillips (PSP) model [1]), takes roughly 3 days for a million time-steps[1]. During design, each simulation of such an ADC can take about 30M time-steps, translating to 3 months of computation to predict what the real circuit would do over 3 seconds. For designers today, this compute time is necessary because every individual device in the circuit that constitutes an ADC is very complex. Many of the physical phenomena captured by the compact models (PSP or Berkeley Short-channel IGFET Model (BSIM) [2], for example) can affect system level behavior.

For any physical device, at the microscopic level, materials, lattices, atoms and electrons have rich interactions and produce complex overall behavior. With continuously shrinking devices, this complexity is now reflected in the macroscopic (system level) properties, like amplifier gains, bandwidths, *etc.*, as well. In order to cope with the increasingly complex behavior of devices, as new physical phenomenon are observed, equations and unknowns are added to existing models. These additions have accumulated over the last several decades of device scaling. Resulting primarily from the need for a single model to suit all application domains, today's models are very bulky and slow.

Table 1 in Section 4, for example, lists the number of calls to elementary arithmetic operations when trying to find the steady state operating point of a CMOS inverter circuit. While searching for an operating point with the PSP model, there are close to 100 Million calls to multiplication and division alone. At the heart of the computational complexity of simulating analog circuits lies the complexity of individual models. At the same time, these models capture a large variety of physical phenomenon and predict the behavior of real devices very well. If it was possible to construct a method to faithfully reproduce the behavior of individual devices while significantly cutting down the cost of evaluating the models, it could have a significant impact on the analog design flow.

Polynomial interpolants are especially suited for this because of the relatively low number of dimensions that have to be dealt with. The first half of this thesis addresses this by building and improving a general 'table-based modeling' framework for approximating device models with polynomial interpolants.

Side effects of model complexity

In many practical scenarios, the 'level of detail' with which a model approximates a device can be different for different operation regions. Region 'X' of operation can be modelled with high fidelity while having a coarser model for another region 'Y'. For example, if one were to design an operational amplifier (*system*) in which transistors (*devices*) are biased near saturation (*'X'*), detailed modeling of the sub-threshold

[1] A single time-step could be anywhere between 1 ns to 1 us, depending on the application.

regime (*'Y'*) of the transistors would lead to an unnecessary increase in both model-development and compute time. Further, the increase in design/simulation time and complexity would offer very little additional insight into the system's operation.

Existing compact models cater to these practical requirements, and often, some operation regimes are modelled in greater detail than others. This is typically done by adding both empirical and physically derived equations inside the model in conditional (*if*, *else*, *etc.*) blocks, or sometimes, using functions like *abs*, *sign*, *etc.*, which provide the same functionality.

Effectively, across operation boundaries, we have different mathematical models which share the same variables. For the model to be physically realistic and mathematically sound, it must be smooth as we transition across the operation regimes. As a result, we add the task of model smoothing across operation regimes to the model developer. Even a model with a few external inputs has internal variables, with respect to all which, the model's equations have to be continuous and differentiable to, at the very least, a few orders across operation boundaries. BSIM for an instance, a common model for MOSFETs, can have up to 12 internal unknowns besides the terminal quantities. For multivariate functions, smoothing all the variables across all operation regimes by hand can quickly become unmanageable.

Simulation algorithms, especially ones based on Newton Raphson (NR)[2], rely on the existence and continuity of derivatives. In Radio Frequency (RF) design, high-order derivatives are directly linked to mixing and harmonics in the response to periodic signals. In Section 7, we perform Harmonic Balance (HB) on an analog amplifier and show that discontinuities in the device model derivatives lead to nonphysical and incorrect simulation results. While automatically fixing model discontinuities is a hard problem, identifying modeling errors in itself can be a very helpful tool for device model developers. The later part of the thesis uses the table-based modeling framework for identifying modeling errors.

Organization of the rest of the thesis

The rest of the thesis is organized as follows: Section 2 details the mathematical preliminaries required, including splines, Barycentric-Lagrange Interpolation (BLI), and Chebyshev series approximation. Section 4 describes the basic framework, called Spline-based Tables for Efficient and Accurate device Models (STEAM) [3], for table-based modeling that this work is based on. Preliminary results with cubic splines are also presented here. In Section 5, we extend STEAM by using Chebyshev polynomials to achieve near machine precision with our table-based approximations, while taking up lesser memory for the look-up tables themselves. This methods is dubbed A LAgrange-interpolant with Chebyshev-samples for Accurately Representing TablE-models (A-LA-CARTE). Section 6 explores the use of Compressed Sensing (CS) for constructing table-based models from a sparse set of samples. This cuts down the cost of evaluating compact models for building table based approximations. Some of

[2]Quiescent Steady State (QSS), AC analysis, transient analysis, harmonic balance, periodic steady-state, envelope simulation, to name a few.

the observations made in Section 5 lead us to use Chebyshev polynomials for not only building table-based models, but also as a diagnostic tool for finding errors in compact models. This is demonstrated in Section 7. Section 8 addresses open questions and future research directions. Finally, Section 9 provides an overview of the software package **STEAM**. Instructions on using provided examples and reproducing main results presented in this report can also be found here.

2 Preliminaries

In this section, we talk about splines, Lagrange interpolants, and Chebyshev sample points. A short introduction of Compressed Sensing (CS) and related transformations is included in Section 6. The MODel SPECification (MODSPEC) modeling API is briefly introduced in Section 4 where it is relevant.

Polynomial interpolation has a very long history, and therefore, a complete background is not possible in a single document. There are a few interesting categories, each of which have distinguishing properties that make them suitable for the problem that we are trying to solve.

2.1 Splines

Consider a function f that takes in a scalar argument x. We are interested in approximating this function, $f(x)$, in a domain $[a, b]$. While we would like our approximation, $\hat{f}(x)$, to be evaluable in $(-\infty, \infty)$, we only want the approximation to be accurate in $[a, b]$. A spline interpolant can be constructed for this purpose.

Splines, in general, are piecewise polynomials, used for interpolating a function from sample values provided at a number of sample points (called *knots*), *e.g.*, $a = x_0 < x_1 < \ldots < x_n = b$. A key feature of spline interpolation is that it uses *local polynomials*, *i.e.*, separate polynomials between pairs of adjacent knots. These are stitched together by enforcing continuity and differentiability at the knots. The local nature of splines also leads to fast computation and localized memory accesses during their evaluation.

More precisely, a spline interpolant is defined as a piecewise polynomial interpolant belonging to the space $\mathcal{S}_{\Delta}^{k,l}([a, b])$, of functions in $\mathcal{C}^{l-1}([a, b])$[3], that are polynomials of degree $\leq k$ locally. Δ defines the set of intervals $\{\Delta_1, \Delta_2, \ldots, \Delta_n\}$[4]. Therefore, within each interval Δ_i, the interpolant is at most a polynomial of degree k. Cubic splines ($\mathcal{S}^{3,3}$), which we use in this work, match sample value, as well as first and second derivatives of the polynomials that are used to approximate the original function in the neighborhood of each knot. This is desirable for smooth modeling. Evaluation of the interpolant at any point is computationally cheap, involving merely a cubic polynomial in the input variable.

2.1.1 Uniqueness

One of the most interesting properties of cubic splines is that for a given set of knots and associated function values, an interpolant that meets all the requirements above is not unique. For each interval Δ_i, we want a cubic polynomial which has 4 unknowns. This gives us a total of $4n$ unknowns.

For the constraints, we have $2n$ constraints for matching the function values at each knot in each interval. Enforcing the continuity of 1^{st} and 2^{nd} derivative at all *interior* knots (*i.e.* knots $1, \ldots, n-1$), gives us an additional $2(n-1)$ constraints. In all, we have $4n$ unknowns and $4n-2$ constraints. Therefore, given a set of knots and

[3]Global degree of smoothness $l-1$.

[4]The interval Δ_i is defined as: $\Delta_i = [x_{i-1}, x_i)$.

sample values at the knots, a cubic spline interpolant is not unique [4]. It can be made so by imposing 2 additional constraints, typically at or near the first and last knot points. So-called *natural splines* and *not-a-knot splines* are two examples of splines that result from slightly different constraints [4]. In Section 4.3, we show that minor changes to natural splines result in an interpolation scheme that intuitively provides better performance in simulation.

2.1.2 Multiple dimensions

In [4], Carl de Boor presented a technique for extending univariate splines to higher dimensions using tensor products:

$$\mathcal{H}_\Delta(\lambda_1, \lambda_2, \ldots, \lambda_N) = \mathcal{S}_{\Delta_1}(\lambda_1) \otimes \mathcal{S}_{\Delta_2}(\lambda_2) \otimes \ldots \otimes \mathcal{S}_{\Delta_N}(\lambda_N), \tag{1}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_N$ are N input variables. The corresponding scalar output[5] is given by $\mathcal{H}_\Delta$. The functions $\mathcal{S}_{\Delta_1}(\lambda_1), \mathcal{S}_{\Delta_2}(\lambda_2), \ldots, \mathcal{S}_{\Delta_N}(\lambda_N)$ are univariate splines in variables $\lambda_1, \lambda_2, \ldots, \lambda_N$, respectively, and $\lambda_i \in \Delta$. The product, given by $\otimes$ in Equation 1, is the tensor product of N univariate splines. For example, the tensor product of two univariate cubic splines $\mathcal{S}_{\Delta_x}(x)$ and $\mathcal{S}_{\Delta_y}(y)$ is the bi-cubic function

$$\begin{aligned} \mathcal{H}(x, y) &= \sum_{j=0}^{3} \sum_{i=0}^{3} c_{ij} x^i y^j, \text{ where} \\ \mathcal{S}_{\Delta_x}(x) &= \text{span}(x^i, i \in \{0, 1, 2, 3\}). \end{aligned} \tag{2}$$

Say we have a function $g(x, y)$ of 2 variables, x and y, that has been sampled over a rectangular grid of $n_x \times n_y$ points. We can approximate $g(x, y)$ with a tensor product spline, $\mathcal{H}$, of the form:

$$\mathcal{H} = C_3(y)x^3 + C_2(y)x^2 + C_1(y)x + C_0(y), \tag{3}$$

where $C_i(y)$ is a cubic function of y. Substituting the expansions for each $C_i(y)$ gives us the general expansion in Equation 2. The form in Equation 3 yields a more tractable implementation. First, we build splines in x for each of the n_y sample points. This gives us n_y splines of the form:

$$\begin{aligned} \mathcal{H}_k &= C_{3,k}x^3 + C_{2,k}x^2 + C_{1,k}x + C_{0,k}, \\ & k \in 0, 1, \ldots, n_y. \end{aligned} \tag{4}$$

The coefficients obtained from these spline $C_{i,j}$, can be treated as samples of the function $C_i(y)$, allowing us to fit a spline over the computed spline coefficients in x. This will give us, in total, $16n_x$ coefficients for each of the n_y coefficients, giving us all the coefficients needed for a bi-cubic expansion. This makes spline evaluation inexpensive and local. However, this is achieved at the cost of more memory since we need to pre-compute and store $16n_xn_y$ coefficients for $n_x \times n_y$ sample points. Tensor-product splines inherit continuity and smoothness of univariate splines, making them well suited for device modeling and circuit simulation.

[5]It is easy to extend this idea to an output that is a vector, however, for the sake of simplicity, we will restrict the exposition to a scalar function. $\mathcal{H}$ can, for example, be the interpolation function representing f in Equation 17.

2.2 Lagrange Interpolation

Continuing the case studied previously in Section 2.1 with Spline interpolants, the value of a single-input, single-output function, $f(x)$, is known at a set of *knots*, labelled $a = x_0 < x_1 < x_2 < \ldots < x_n = b$. The Lagrange interpolation formula [5], interpolates between the known function values using the polynomial expression:

$$L(x) = \sum_i \frac{f(x_i) \prod_{j \neq i} (x - x_j)}{\prod_{j \neq i} x_i - x_j}. \tag{5}$$

2.3 Barycentric-Lagrange Interpolation

Barycentric Lagrange Interpolation formula, as described in [5], is obtained by a small tweak of the Lagrange interpolation formula. The first point to note is that the polynomials, $p_i(x) = \prod_{j \neq i} (x - x_j)$, can be be expressed in a simpler, non-polynomial form as:

$$p_i(x) = \frac{l(x)}{x - x_i}, \text{ where } l(x) = \prod_j (x - x_j). \tag{6}$$

Applying this modification to the Lagrange Interpolation formula in Equation 5 gives us the *Modified Lagrange Interpolant*, which can be expressed as:

$$L(x) = l(x) \sum_i \frac{f(x_i) w_i}{(x - x_i)}, \text{ where } w_i = \frac{1}{\prod_{j \neq i} (x_i - x_j)} \tag{7}$$

Interpolating the constant function $f(x) = 1$, again using the modified Lagrange interpolant, we obtain an interpolant, $L_{trivial}(x)$, as follows:

$$L_{trivial}(x) = l(x) \sum_i \frac{w_i}{(x - x_i)}. \tag{8}$$

This is essentially obtained by substituting $f(x_i) = 1, \forall i$. Since $L_{trivial}(x)$ is a polynomial of degree n, and takes the value 1 at $n + 1$ points, it must be identical to 1 for all x. Therefore, one can obtain a computationally efficient formula for computing the interpolant by dividing the Modified Lagrange formula in Equation 7 with the expression in Equation 8 to obtain

$$B(x) = \sum_i \frac{f(x_i) w_i}{(x - x_i)} \Big/ \sum_i \frac{w_i}{(x - x_i)}. \tag{9}$$

2.4 Chebyshev Polynomials

Chebyshev polynomials are a family of polynomials defined on the domain $[-1, 1]$. This family has 4 main branches, named just by a number. We will primarily be using Chebyshev polynomials of 1^{st} and 2^{nd} kind.

Chebyshev Polynomials of the 1^{st} kind:

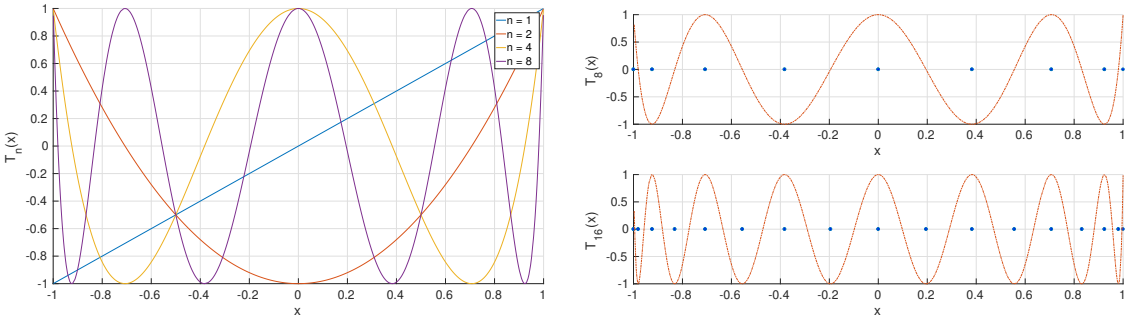
$$T_n(x) = \cos\left(n \cdot \cos^{-1}(x)\right). \tag{10}$$

Chebyshev Polynomials of the 2^{nd} kind:

$$U_n(x) = \cos\left(\left(n + \frac{1}{2}\right) \cdot \cos^{-1}(x)\right). \tag{11}$$

Despite the presence of *cos* and cos^{-1} in their description, for integer values of n, these are polynomials. Figure 1(a) shows Chebyshev polynomials of the first kind for orders $n = [1, 2, 4, 8]$. For a more detailed description of the families and their individual properties like orthogonality, nesting of roots and extremas *etc.*, readers are encouraged to refer to [6].

2.5 Chebyshev Points



(a) $T_n(x)$ (b) Chebyshev Points

Figure 1: Chebyshev polynomials and Chebyshev points.

Chebyshev points are defined as the roots or extremas of Chebyshev polynomials mentioned above. As an illustrative example, for Chebyshev polynomials of the first kind in Equation 10, the roots are given by:

$$\begin{aligned} n \cdot \cos^{-1}(r_i) &= \left(i + \frac{1}{2}\right)\pi, \\ r_i &= \cos\left(\left(i + \frac{1}{2}\right)\frac{\pi}{n}\right). \end{aligned} \tag{12}$$

These are called Chebyshev points of the first kind. Similarly, Chebyshev points of the second kind of order n are the extremas of the polynomial $T_n(x)$. These are given by the solutions of $T_n(x) = \pm 1$.

$$\begin{aligned} n \cdot \cos^{-1}(e_i) &= i \cdot \pi, \\ e_i &= \cos\left(\frac{i \cdot \pi}{n}\right). \end{aligned} \tag{13}$$

Figure 1(b) shows Chebyshev points for $T_8(x)$ and $T_{16}(x)$ respectively. Observe that the extremas of $T_{16}(x)$ are nested in the extremas of the previous polynomial $T_8(x)$. Chebyshev points of second kind are used in [7] (and the tool described therein, called

CHEBFUN) because the set of Chebyshev points of second kind of order n is nested in the set of Chebyshev points of order $2n$. This makes incrementing the sample values of a function at the Chebyshev points computationally efficient, as increasing the order from n to $2n$ requires evaluation of the function at only n points.

Besides, using Chebyshev points for interpolation with Barycentric-Lagrange Interpolation (BLI) is numerically stable. The weights w_i, described in Equation 7 depend on the choice of sample points and are vital for the overall stability of the interpolant. As noted in [5], these weights for a sampling domain of $[-1, 1]$[6], for a set of uniform sample points are given by:

$$w_i = (-1)^i \binom{n}{i}. \tag{14}$$

While the smallest absolute value of a weight is that of $w_0 = 1$, the maximum value grows exponentially[7] with n. With Chebyshev points however, these weights are given by:

$$w_i = (-1)^i \delta_i, \text{ where } \delta_i = \begin{cases} 0.5 & x = 0, \text{ or } x = n \\ 1 & \text{otherwise} \end{cases} \tag{15}$$

2.6 Chebyshev Series

Chebyshev polynomials discussed above in Section 2.4 can be put together in the form of a weighted series:

$$S_n(x) = \sum_{k=0}^{n} c_k \cdot T_k(x). \tag{16}$$

Such a series is called a Chebyshev series. Chebyshev polynomials $[T_0(x), \ldots, T_n(x)]$ form a basis for the space of polynomials with degree $\leq n$, and therefore, Chebyshev series can be used to approximate any function by a polynomial of degree n. Chebyshev polynomials themselves have similar characteristic to the elements of Discrete Cosine Transform (DCT) basis, which can also be intuitively observed from Figure 1(a), where polynomials of higher order seem to capture *high frequencies*. This results in rapidly dropping Chebyshev series coefficients for smooth functions, similar to what is observed for DCT. The advantage of this representation over conventional DCT in the present context is that Chebyshev series is identical to the interpolant that we construct.

[6] for any other sampling domain, the weights are scaled by a constant factor

[7] These values grow so fast that straight-forward implementations of w_i run into numerical overflow even for moderate values of n used in the experiments in the upcoming sections.

3 Previous Work

Since the early 1980s, various methods have been proposed for incorporating table-based device modeling into circuit simulation. Owing to their popularity and complexity, MOS models have often been the central theme for publications on table-based device modeling.

In 1983, CAzM[8], a macromodeling simulator, was developed. CAzM created a 'macromodel' of a sub-circuit that captured its steady state behavior. In order to build such a macromodel, the terminal $i - v$ characteristics of the sub-circuit were obtained for a set of terminal inputs (typically bias voltages). To analyze the steady-state of the sub-circuit at any give input, aforementioned $i - v$ characteristics were interpolated. Later, CAzM was updated to include a charge model by building similar tables for $q - v$ characteristics at the terminal nodes [9]. In [10, 11], splines were proposed to interpolate branch currents for 4-terminal MOSFETs. Raw device evaluation speedups up to 3× were reported in [10].

As discussed in [3], while measurement/characterization of terminal currents alone is sufficient for DC analysis for most circuits, characterizing charges at terminals alone leads to an incorrect approximation of a model. One of the main reasons why more elaborate and accurate table-based models have not been built is memory requirement. Building internal nodes and implicit equations into a table-based model requires look-up tables with more dimensions, and therefore larger memory consumption. [9], the update to CAzM in 1992 mentioned above, uses a total of 540 sample points to build tables for a single 4-terminal MOSFET. As can be seen in Section 4, such a small number of sample points is insufficient for the accuracy requirements for analog circuit simulation today.

Later work in this direction [12, 13, 14], follows the same paradigm, and is unable to advocate the use of table-based models for accelerating circuit simulation sufficiently. Part of this can also be attributed to the lack of a clean, modular platform for device modeling. In [12] a table-lookup scheme is implemented in SPICE. Such an implementation is difficult as algorithms and device models are not well-separated in SPICE. [12] reports a speedup of 1.4× in the SPICE implementation using linear interpolation. Their scheme suffers from significant modelling errors (up to ∼40%), largely because of the crude, non-smooth interpolation scheme(s) used. Using more accurate interpolation methods in [12] results in a slowdown over the analytical compact models that table-based models are derived from.

Meanwhile, research on table-based modeling progressed in very exciting directions. In [15], the authors claim that the analytical models available at the time (including early Berkeley Short-channel IGFET Model (BSIM)) were incapable of handling several short-channel effects, which motivated them to build table-based models to account for these inaccuracies. The dynamic component of the model is handled by tabulating the drain conductance g_{ds} as a function of the gate and drain voltages. Although such an approach is not a paradigm shift in the correctness of table-based models, it does point to the fact that as new devices are developed, compact models do not catch up rapidly to meet the accuracy requirements. Eventually, patches are applied for considering the new effects making model evaluation slow and complex.

4 STEAM

As discussed previously in Section 3, one of the factors holding down table-based device modeling is the lack of a modular implementation of a circuit simulator, and also a flexible, open modeling framework in which device models can be expressed. The de-facto industry standard for expressing compact models is, in fact, a non executable language, called Verilog-A. The inability to execute a model makes it very difficult to isolate it from the simulator. Model equations, unknowns, and functions have to indeed be inferred from a Verilog-A description making it difficult to have a simulator independent method for converting analytical compact models in to table-based models. Presently, a flexible framework for constructing table-lookup models from existing compact models seems lacking.

The MODel SPECification (MODSPEC) modeling API [16] offers both transparency into the model structure and is also executable, making it independent of any simulator. A MODSPEC description of a compact model is supported in two open source circuit simulators: Modeling and Algorithm Prototyping Platform (MAPP), and Xyce.

In [3], we built a basic framework around MODSPEC for approximating any compact model with a polynomial interpolant. This was dubbed Spline-based Tables for Efficient and Accurate device Models (STEAM). Here, we explain the framework and later sections (Sections 5, 6, and 7) build on this framework to resolve issues specific to speed, accuracy, and memory requirements.

While the premise of STEAM in [3] is the MODSPEC modeling API, the framework only demands that the characteristics of the Device of Interest (DoI) be expressible as a set of functions of *some* inputs. MODSPEC is only one such example.

4.1 Functional view of compact models

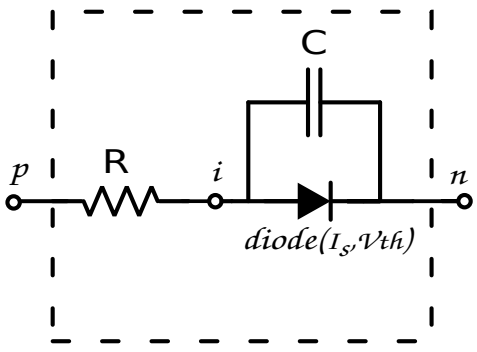


Figure 2: Schematic for a diode model.

In order to express a device model, MODSPEC uses a compact set of up to 4 functions, each having a physical meaning for the device behavior. A detailed description can be found in [16]. We provide an example here to highlight the API's main aspects. The equation governing the dynamics of the device in Figure 2 is:

$$\frac{d}{dt}(Cv_{i,n}) + diode(v_{i,n}) + \frac{v_{i,p}}{R} = 0. \tag{17}$$

Here, the ideal diode equation is given as:

$$diode(v_{i,n}) = I_s \cdot (e^{\frac{v_{i,n}}{V_{th}}} - 1). \tag{18}$$

For the simple device being discussed here, the *functions* that we are considering are:

$$\begin{aligned} q(v_{i,n}, v_{p,i}) &= Cv_{i,n}, \text{ and,} \\ f(v_{i,n}, v_{p,i}) &= diode(v_{i,n}) + \frac{v_{i,p}}{R}. \end{aligned} \tag{19}$$

Model	Calls	(*)	(+)	(-)	(/)	(**)
PSP	7840	85679787	37689915	18184544	11983440	5354720
STEAM (PSP)	7840	412731	653472	223129	156825	0
BSIM	5810	10640142	4898368	2062882	2028271	888930
STEAM (BSIM)	5810	368643	514746	174652	125516	0

Table 1: Comparing the total number of mathematical operations for different transistor models in DC analysis.

More complex compact models, like Berkeley Short-channel IGFET Model (BSIM), and Penn-State Phillips (PSP) models, involve many more arithmetic operations and calls to functions like *exp*, *log*, *etc*. Table 1 shows the counts for simple arithmetic operations for different compact models, highlighting the complexity of models like BSIM and PSP, which are commonplace in industrial circuit simulation. At the same time, we also show the arithmetic operations required for a table-based model constructed from STEAM using the respective compact models. The approximation constructed with STEAM requires roughly $100\times$ fewer operations, resulting in a similar speedup over the conventional models.

Since STEAM relies on splines, the operation count doesn't increase with the accuracy of the model. A slight increase can be seen if the sample points used for constructing the interpolant are chosen to be non-uniformly distributed. This would involve a search in the set of sample points for the appropriate interval which increases as $log(n)$, where n sample points are chosen for each input dimension. For structured sample points, like Chebyshev points, this can still be avoided by applying a mapping to uniform points (See Section 2.5 for details.).

STEAM is a very simple idea that the multi-variate functions that describe a compact model, can be replaced with cubic splines for better speed. As can be noted from Table 1, while the original functions that we start with are complex, the substituting polynomials are far simpler at the cost of memory. One could possibly use higher order splines, however, cubic splines already offer continuous first and second derivatives which are sufficient for a large number of applications.

4.2 Derivatives

Simulation algorithms often require not just the device evaluations, but also the derivatives of the evaluated values with respect to the inputs. Newton Raphson (NR), for instance, a common algorithm used for finding a zero of an algebraic expression requires the derivative of the expression with respect to the input arguments. Referring to the MODSPEC terminology, it would require the derivatives of f and q functions with respect to the inputs. Since we are approximating f and q with cubic splines, we would also need the derivative(s) of the spline interpolant.

Recall that in 2 dimensions, the spline expression is:

$$\mathcal{H} = C_3(y)x^3 + C_2(y)x^2 + C_1(y)x + C_0(y). \tag{20}$$

The partial derivatives for this are given by:

$$\begin{aligned}\frac{\partial \mathcal{H}(x,y)}{\partial x} &= 3C_3(y)x^2 + 2C_2(y)x + C_1(y), \text{ and} \\ \frac{\partial \mathcal{H}(x,y)}{\partial y} &= \frac{dC_3(y)}{dy}x^3 + \frac{dC_2(y)}{dy}x + \frac{dC_1(y)}{dy}x + \frac{dC_0(y)}{dy}.\end{aligned} \tag{21}$$

These can easily be computed using the pre-evaluated spline coefficients and the ability to evaluate cubic expressions.

4.3 Extrapolation

Besides the requirement for derivatives for the device functions, methods like NR used for solving circuit equations often sample the devices at infeasible input values. If the circuits are constructed correctly, the final solution is usually in the input range that we would expect the device to operate, but intermediate computation steps can lead to device evaluation outside this range (say $1000V$ for a forward biased diode.). The value and derivative returned by the model for these inputs determines how quickly the algorithm will return to a feasible value, and therefore, converge to the solution.

This implies that the table-based approximation should at least return 'reasonable' values and derivatives outside the region which it was built for. If the model can be made to return values that help with NR convergence, it can be useful for reducing the overall simulation time. As discussed in Section 2.1, for a give set of sample values, the spline interpolant is not unique. In 1 dimension, for example, we have 2 Degrees of Freedom (DoF) to describe the interpolant. These can be used to set the terminal slope, as demonstrated in Figure 3. The ability to adjust terminal slopes allows hand-tuning for NR convergence. We call this method passive extrapolation because as

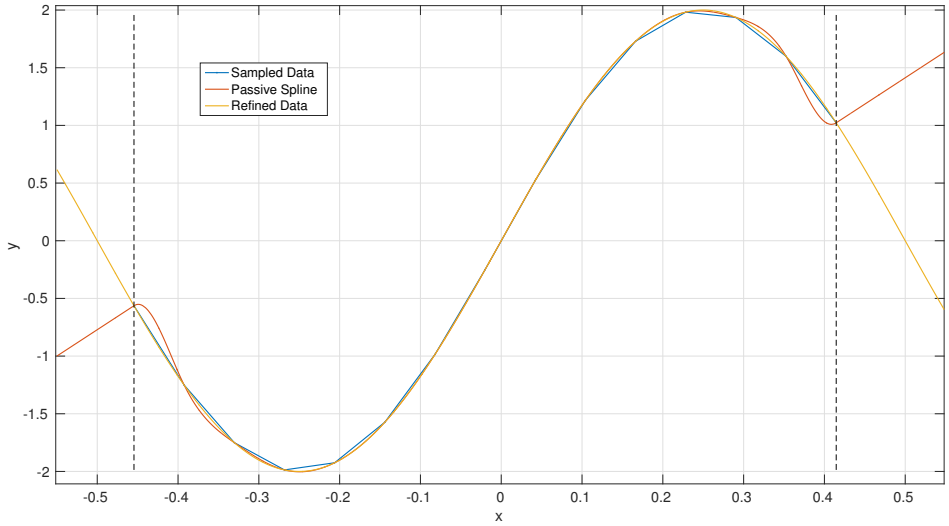
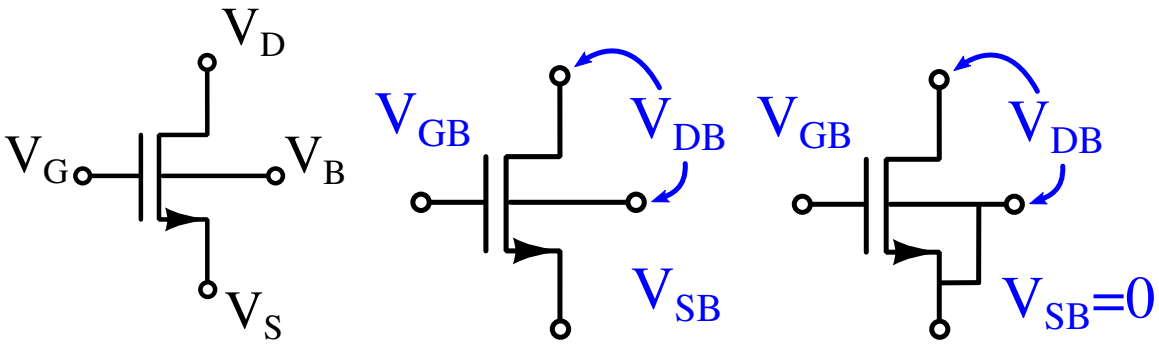


Figure 3: Illustrating passive extrapolation on a test function.

shown in Figure 3, this can be used to make any set of sample values behave like a passive device.

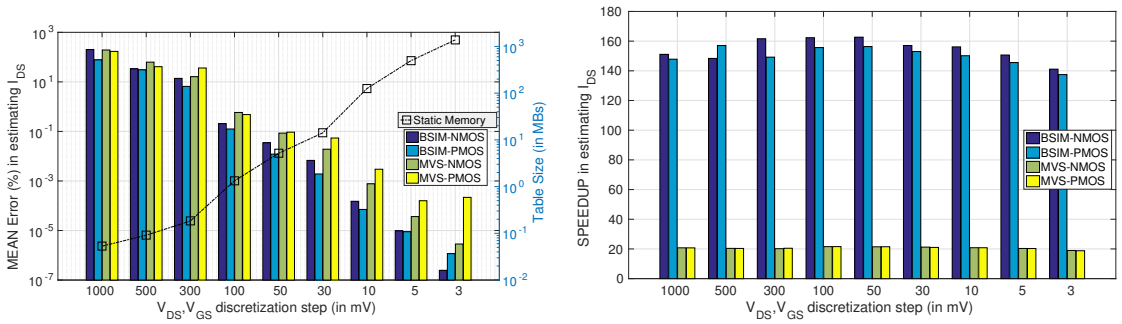
4.4 Device evaluation



(a) node-based model (b) branch-based model (c) 2-Dimensional model

Figure 4: Schematic of a transistor model.

In order to assess our method, given a compact model like BSIM or MIT Virtual Source (MVS), we construct an approximation with STEAM. Figure 4 shows the schematic of the transistor model on which STEAM is being applied. While internal nodes are vital for the approximation to be correct, at the moment, we will ignore the internal nodes in these models for simplicity. For the schematic shown, voltages v_{db}, v_{sb}, and v_{gb} are the inputs, and currents i_{db}, i_{sb}, and i_{gb} are the outputs.[8] In order to further simplify the table-based model and to visualize the device functions, for the remainder of this section, we will restrict ourselves to the 3-terminal (2 input) model shown in Figure 4(c).



(a) Accuracy (b) Speedup

Figure 5: Comparing the accuracy and speedup in approximating compact models with cubic splines (STEAM [3]).

In order to build an approximation of the model in Figure 4(c) with STEAM, we evaluate the original compact model's MODSPEC description on a Cartesian grid in the input space. For example, we could choose the sets $S_{v_{db}} = \{-1.0, -0.7, -0.3, 0.3, 0.7, 1.0\}$, and $S_{v_{gb}} = \{-1.0, -0.5, -0.3, 0.1, 0.7, 1.0\}$. We would then evaluate the original compact model on $S_{v_{db}} \times S_{v_{gb}}$, and use the evaluated values for the MODSPEC functions

[8]Branch currents and branch voltages are simply defined as the differences of the respective node quantities. For example, the current i_{db} in Figure 4(b) is given by $i_d - i_b$ in Figure 4(a).

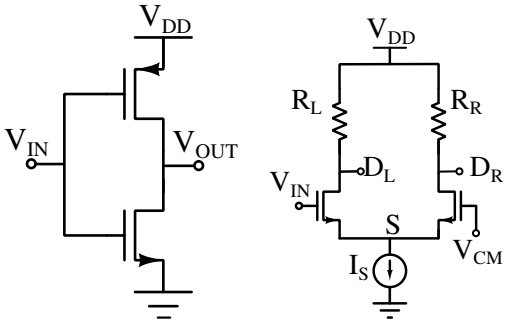
$f(v_{db}, v_{gb})$, and $q(v_{db}, v_{gb})$ to fit a 2-dimensional cubic spline. An interpolant that can reconstruct all the MODSPEC functions at any given input value can act as a complete substitute for the original model in a circuit.

Having constructed a STEAM approximation of the device model, we can analyze its accuracy by comparing the STEAM model's outputs at inputs other than the sample values used to build the model. Figure 5 shows the accuracy and speedup results for such device evaluation alone. In order to build the model in Figure 5, the sets $S_{v_{db}}$, and $S_{v_{gb}}$ are all constructed by simply splitting a domain $[-1, 1]V$ uniformly into a number of pieces. In both Figures 5(a), and 5(b), the x-axis shows the *step size* for the uniform split, whereas the y-axis shows the mean relative error and speedup in computing f for the device model.

For larger discretization step sizes, the error is high. For example, if both input axes were to be discretized with a 100mV discretization step for constructing the model, the mean error would be roughly 0.1%.[9] At the same time, lowering the discretization step to 3mV for all the three inputs reduces the mean error to 10^{-7}% for the BSIM model.

The speedup, as shown in Figure 5(b), remains largely unaffected by the discretization step. This is primarily because the computation needed to evaluate a STEAM model, for any discretization step, is indeed the same: the evaluation of a bi-cubic polynomial. As the discretization step drops, the memory requirements increase significantly and this has an impact on the memory efficiency.

4.5 Accuracy, Memory and Speedup in analyses

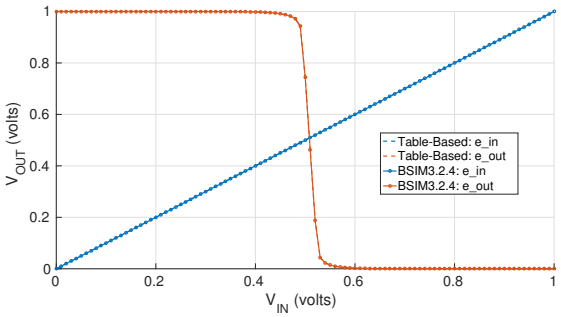


(a) inverter (b) differential pair

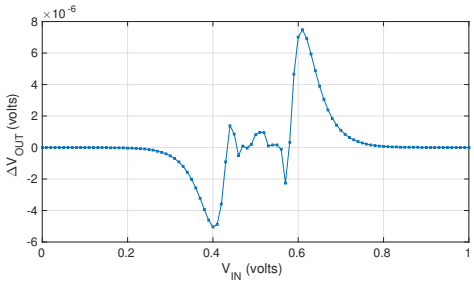
Figure 7: Circuits used for various analyses.

Both the accuracy figures, and speedup obtained in evaluating the device models suggest that the look-up table approach proposed in STEAM could be used to improve the runtime for several analysis algorithms. The most common among these are Quiescent Steady State (QSS), transient analysis, and AC analysis. Later in the thesis, we also explore Harmonic Balance (HB), a mixed time-frequency domain analysis used in Radio Frequency (RF) design. For each of the analysis mentioned next, we start with a compact model (BSIM, and MVS), and construct its STEAM approximation. We then connect this model to the rest of the circuit and perform the analysis. End results of the analysis are obtained for both the models and compared for accuracy. The overall time taken by the analysis is also compared later in Figure 8(b).

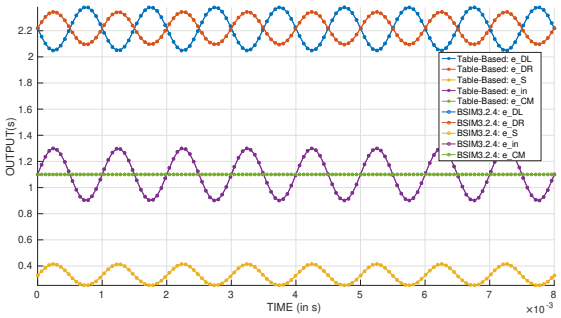
[9]The error is measured by evaluating both the original model and the STEAM approximation at $\sim 100K$ randomly chosen points.



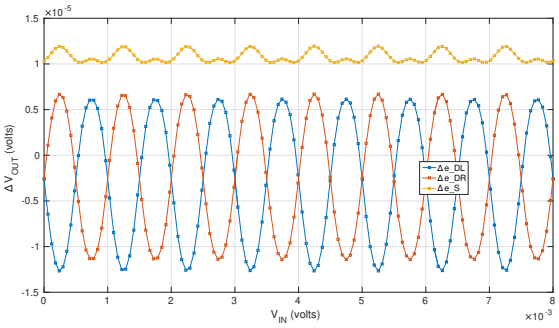
(a) DC Analysis



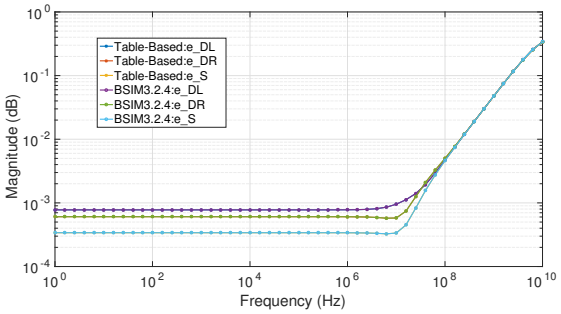
(b) Error in DC



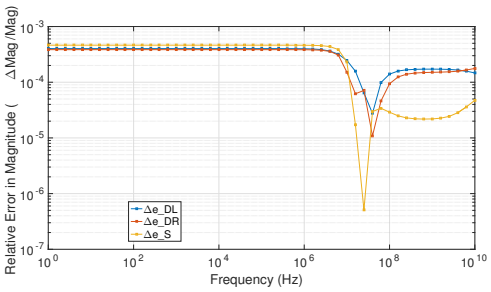
(c) Transient Analysis



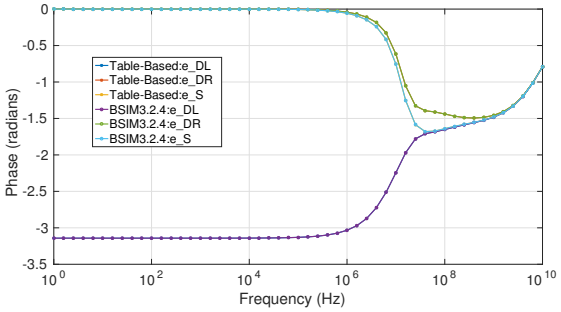
(d) Error in Transient



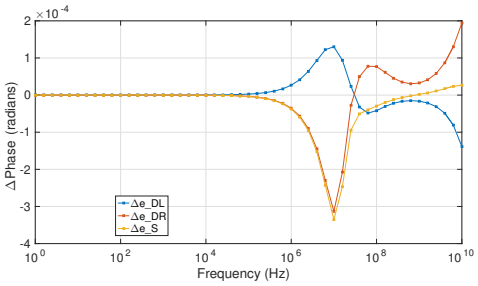
(e) AC Analysis - Magnitude



(f) Error in AC Analysis (Magnitude)



(g) AC Analysis - Phase



(h) Error in AC Analysis (Phase)

Figure 6: Comparing analysis and respective errors in using STEAM approximation of compact models.

4.5.1 QSS

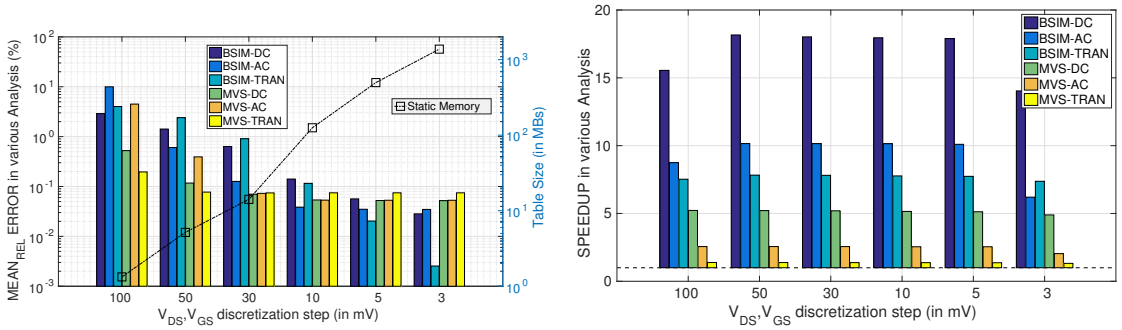
QSS is used to obtain the steady state of a circuit or a system when the inputs are all held at a steady value. In order to compare the accuracy of the STEAM approximation of BSIM and MVS models, we sweep a single input for a circuit, and compare the operating state obtained from the two models. The results for a CMOS inverter (circuit schematic shown in Figure 7(a)) can be seen in Figure 6(a). The error between the simulation output of the original compact model and the table-based approximation can be seen in Figure 6(b). As the input is swept, and the output voltage swings from $1V$ to $0V$, the error between the two models is less than $8\mu V$.

4.5.2 Transient Analysis

For transient analysis, the input v_{in} in the circuits in Figure 7, is set to be a sinusoid. The output waveform is recorded at a set of nodes in the circuit and compared for accuracy. Figures 6(c), and 6(d) show the outcome and the error between the two models respectively. Here too, the error at the output nodes (D_L, and D_R) is less than $10\mu V$.

4.5.3 AC Analysis

Circuits are analyzed with AC analysis by fixing the input v_{in} at a constant bias and analyzing the small signal response for the two models. Both the magnitude and phase responses, as well as the errors in them resulting from table-based approximations are shown in Figures 6(e)-6(h).



(a) Approximation Error

(b) Speedup

Figure 8: Aggregate performance of compact models approximated with STEAM.

The aggregate error and speedup have been compiled in Figure 8, showing that the error between a compact model and its table-based approximation can be significantly reduced at the cost of memory. For high accuracy, memory requirements seem high, but not necessarily prohibitive. This is especially true for simulations involving digital circuits where the number of distinct devices (and therefore tables) is relatively low. As seen in Figure 8(b), for almost any choice of discretization step, a healthy amount of speedup can be achieved over compact models. Also, the speedup that can be achieved

with STEAM is qualitatively independent of the accuracy. These numbers were taken by averaging over several trials in MATLAB by clearing the function cache before each trial. This was done to avoid interference from Just in Time (JIT) compilation of code which seems to affect the compact model and table-based model differently.

5 Is accuracy a concern?

Despite the apparent advantages of table-based models demonstrated in Section 4 and some prior publications [8, 12, 13, 14], Electronic Design Automation (EDA) circles seem to have refrained from using table-based device models for industrial applications. Device physicists often include look-up tables to augment existing models for a variety of reasons. For large scale simulations, some arguments have held this category of methods back:

1. Table-based models are not accurate enough to be applicable in analog design, especially with Radio Frequency (RF) analysis algorithms which demand a high level of accuracy,

2. Memory requirements for high accuracy prohibit the use of higher number of dimensions which are needed to include internal nodes, model parameters *etc.*

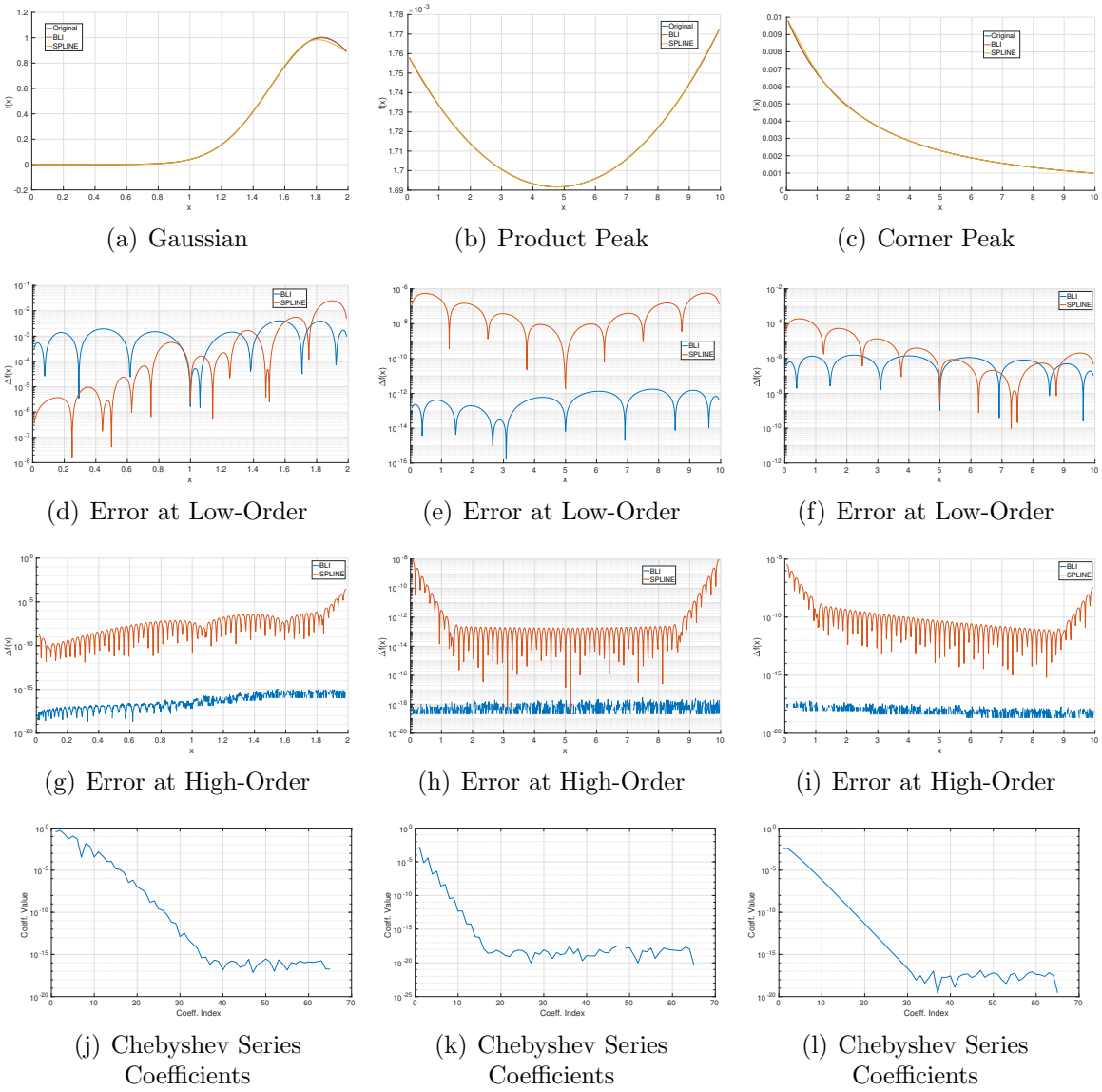
With Spline-based Tables for Efficient and Accurate device Models (STEAM), in Section 4, we demonstrated the promise of performance improvement that could be achieved by using a table-based modeling approach. However, the memory-accuracy trade-off discussed in Section 4.5 on the basis of the empirical data for various analyses algorithms does not adequately answer the concerns raised above that have held table-based models from being accepted in industry. Specifically, for a model accuracy of about 8 decimal digits of precision, using STEAM, we need roughly 1GB of storage for the tables (see Figure 5(a)) for a model with 2 inputs. The current section addresses this issue. We propose the use of piecewise Chebyshev polynomials that significantly cuts down the memory requirements for a given level of accuracy, allowing the use of models with higher input dimensions. Alternatively, this section attempts to answer the question: **"Is it practical to have a table-based approximation that is indistinguishable from the original compact model?"**

In order to answer this question, we first take the best known general-purpose polynomial interpolant, Barycentric-Lagrange Interpolation (BLI) constructed with Chebyshev sample points, and compare it against more commonly used interpolants like cubic splines over a range of test functions. This helps us identify the strengths and weaknesses of BLI. Then, we apply this interpolant to device model functions, discovering some of the details that are essential to adapt BLI to device functions.

This method is dubbed A LAgrange-interpolant with Chebyshev-samples for Accurately Representing TablE-models (A-LA-CARTE). With it, we end up with approximation of a compact model based on BLI that is numerically indistinguishable from the original compact model. More importantly, this can be done while consuming lower memory than the best results obtained with splines earlier but at the cost of a lower performance improvement. While for the same number of sample points, A-LA-CARTE already produces 16× fewer coefficients to be stored, Figure 11 will show that in 2-dimensions, BLI yields better accuracy with 4× fewer sample points. This allows us a total of 64× memory compression while obtaining better accuracy that STEAM.

5.1 Accuracy comparison between Splines and BLI

For a fixed number of sample points, let us compare the accuracy of cubic splines and BLI in approximating a function. Besides the kind of interpolant used (splines *vs.* BLI), the choice of sample points is important. In that regard, we choose between uniform points and Chebyshev points (see Section 2.5 for details). There could in fact be a very large range of options to choose the sampling points from, but there is evidence, especially from polynomial interpolation research packages like CHEBFUN, suggesting that Chebyshev points are a good choice [7, 17, 18, 5]. Uniform points, on the other hand, appeal to the intuition and have a fast and clean implementation.



(a) Gaussian (b) Product Peak (c) Corner Peak

(d) Error at Low-Order (e) Error at Low-Order (f) Error at Low-Order

(g) Error at High-Order (h) Error at High-Order (i) Error at High-Order

(j) Chebyshev Series Coefficients (k) Chebyshev Series Coefficients (l) Chebyshev Series Coefficients

Figure 9: Comparing BLI with Spline for interpolation accuracy and its relation with Chebyshev Series representation.

Figure 9 compares the performance of BLI and Spline interpolants for three smooth

functions, called Gaussian, Product Peak, and Corner Peak. These functions were used in testing in the accuracy of polynomial interpolants in [19]. Plots of the test functions for a random choice of function parameters have been added in Figure 9. Let us look at the Gaussian function for an illustrative example.

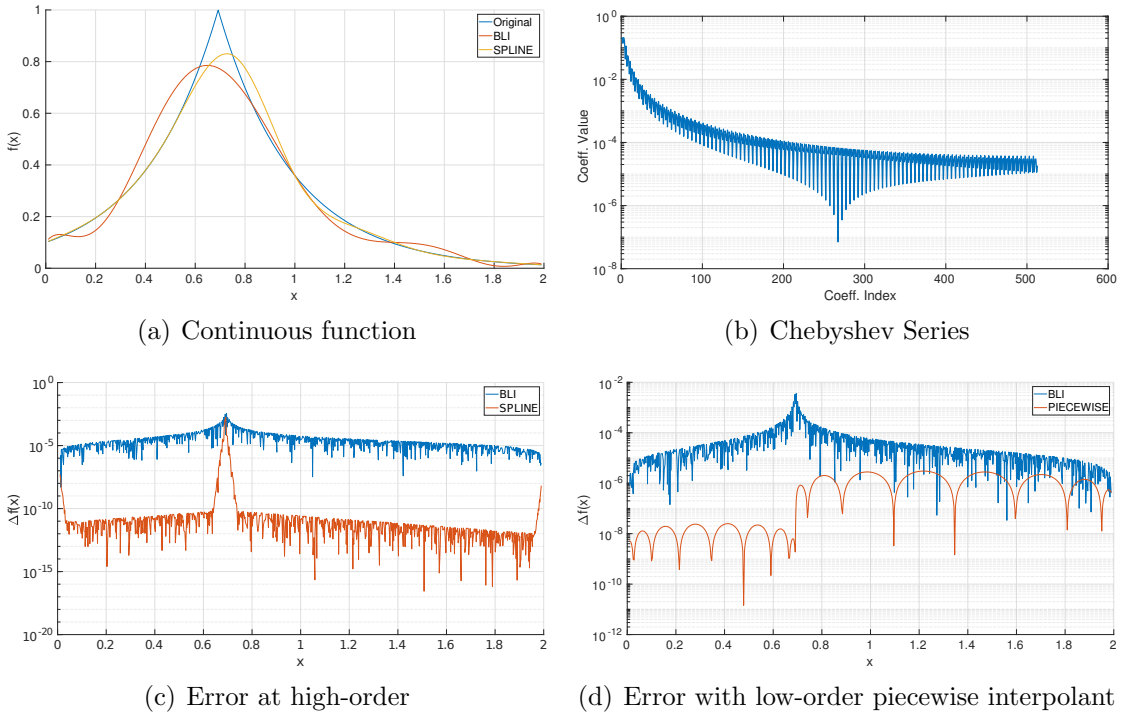
Figure 9(a) shows the plot of the function and an overlaid plot of BLI and Spline interpolants constructed with 9 sample points. The 'eyeball metric' or the visual accuracy of interpolants is limited to 2 decimal digits, and therefore, we compare the error between the original function and the interpolant(s) at a set of points that are randomly chosen, but restricted to be within the sampling domain (in this case [0,2]). Figure 9(d) shows the error in approximating the Gaussian function with BLI and splines, when both of these are constructed using 9 sample points. The accuracy with very few sample points is similar for the two interpolants.

As described in Section 2.6, a Chebyshev series representation gives the approximation of a function in the basis of orthogonal polynomials. Chebyshev series coefficients are directly related to the accuracy of an approximation using BLI with Chebyshev sample points. More precisely, the error in using an interpolant of degree n at any point is bounded by the sum of the Chebyshev series coefficients with indices $> n$.

Figure 9(j) shows the first 65 Chebyshev coefficients of the Gaussian function in Figure 9(a). One can notice that after about 40 indices, Chebyshev series coefficients are smaller than the leading (largest) Chebyshev coefficient by a factor of $\sim 10^{-17}$. Therefore, a BLI constructed with 40 Chebyshev sample points or more should be numerically indistinguishable from the original function. This can be seen in Figure 9(g), where the error is reported between the original Gaussian function and Spline/BLI interpolants constructed with 65 sample points. While the error in spline interpolants drops to $\sim 10^{-10}$, BLI with Chebyshev sample points approximates the function to numerical precision. Uniform points are chosen in case of Splines. Choosing Chebyshev points with the spline interpolant was observed to not affect the interpolation accuracy.

5.2 Interpolating non-smooth functions

As mentioned earlier in Section 1, in practice, many device model functions are not as smooth and well behaved as a Gaussian, exponential, *etc.* Instead, they are written with conditional statements, *sign*, *sqrt*. *etc.*, making them discontinuous, sometimes in value or first derivative(s), and more frequently in higher order derivatives. Here, we summarize one of the most important aspects of using polynomial interpolants for approximating such discontinuous functions. Figure 10(a) shows a function with a discontinuity in the first derivative to illustrate this point. The Chebyshev series in Figure 10(b) seems to suggest that we need over 500 sample points to achieve even 6 decimal digits of accuracy. This is in fact verified by Figure 10(c), which shows the interpolation error in interpolants constructed with 513 sample points. However, if we break the domain into 2 independent domains and construct an interpolant for each of the domains with just 9 sample points each (a total of 18 sample points), then we can achieve a lower error than a single interpolant with 513 sample points across the entire domain (See Figure 10(d)).



(a) Continuous function

(b) Chebyshev Series

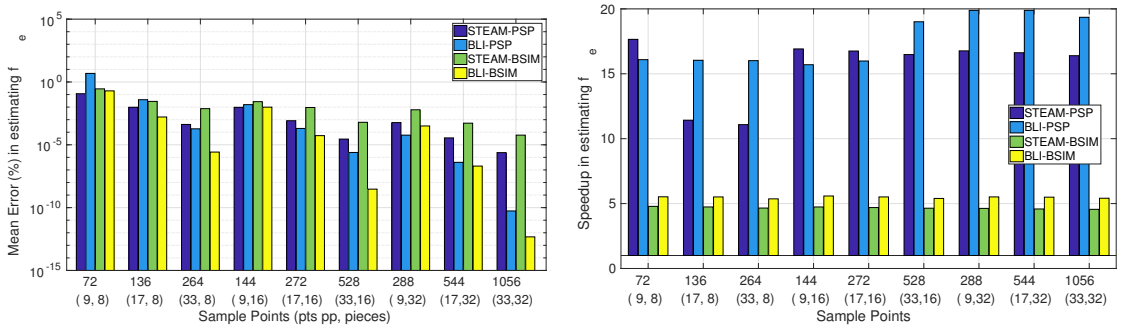
(c) Error at high-order

(d) Error with low-order piecewise interpolant

Figure 10: Comparing single-piece interpolation with BLI for a non-smooth function.

5.3 Machine precision in 2 Dimensions

We implemented BLI with Chebyshev sample point in Modeling and Algorithm Prototyping Platform (MAPP)[20]. Unlike splines, generating a BLI interpolant doesn't require the solution to a linear system of equations, making the generation of the interpolant practically insignificant. It is important, however, to have extrapolation for BLI as Chebyshev polynomials are very unstable outside the domain $[-1, 1]$. First, the desired domain has to be shifted and scaled so that it maps to $[-1, 1]$. We are also forced to use linear extrapolation beyond the original sampling domain to maintain continuity and differentiability at the extrapolation boundary. Listing 1 shows the



(a) Error

(b) Speedup

Figure 11: Comparing the performance of STEAM and A-LA-CARTE.

code for 2-dimensional BLI that we have written for our simulations. One can see that extending BLI to multiple dimensions is highly compact, requiring just 3 calls to the 1D interpolant for 2 dimensions for evaluating a multi-dimensional function as well as its derivatives.

Listing 1: Matlab Code for BLI expressions and derivatives in 2 Dimensions using a 1D interpolant

```
% CLASS BLI2/Public Method
function [val, der] = compute(Obj, x_in)
% Main function for evaluating a 2D BLI expression using
% a Tensor-Product decomposition of 1D BLI interpolants.
%
% INPUTS:
%   - Obj, an instance of BLI2 class. It has 2 important
%     members, BLI_d1 and BLI_d2, 1D interpolants for the
%     two individual dimensions
%
%   - x_in, a vector of size 2x1 comprising of the query
%     point of the form [xq; yq]
%
% OUTPUTS:
%   - val, a vector of size Obj.op_dims, which can in
%     general be a multi-dimensional matrix
%
%   - der, the derivative of 'val' with respect to the
%     two input variables

    % For better readability, let us call
    xq = x_in(1);
    yq = x_in(2);

    % Initializing the derivative for a vector function
    der = zeros([Obj.op_dims, Obj.in_dims]);

    % Computing the 1D BLI interpolant: fx(y)
    [fx_of_y, dfx_of_y_dy] = Obj.BLI_d2.compute(yq);

    % Using interpolated values fx(y) to evaluate:
    %   - B(x, y)
    %   - dB(x,y)/dx
    [val, der(:,1)] = Obj.BLI_d1.compute(xq, fx_of_y);
    % BLI interpolant should be constructed in a way that
    % can either use the stored function values at sample
    % points as in the first call, or use supplied
    % function values for the interpolation (as here)

    % Substituting the values dfx(y)/dy in the BLI
    % expression gives us, as a 'val':
    %   - dB(x,y)/dy
    der(:,2) = Obj.BLI_d1.compute(xq, dfx_of_y_dy);
end
```

Let us extend the use of Chebyshev polynomials to device model functions, and compare it with the existing framework in STEAM. We want to compare the accuracy, memory, and speedup in approximating the core device functions for a MODel SPECification (MODSPEC) model in 2 dimensions.

In Section 5.2, it was shown that if a function being approximated with BLI has a discontinuity in value or derivative, the accuracy drops for a given number of sample points. Currently, we want to *reproduce* the response of a given device model as accurately as possible. This means that if there is a discontinuity in the model, we

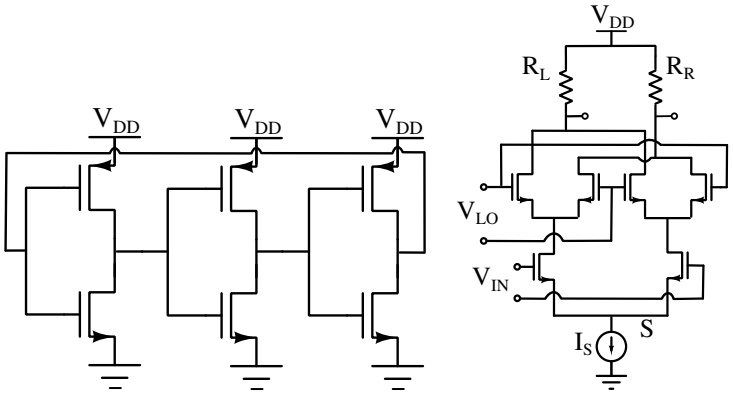
will reproduce it as well. Later, in Section 7, we use these observations to find model discontinuities.

One of the ways of replicating a compact model's response 'accurately' is to split the domain of interest into multiple pieces and fit a separate interpolant to each piece. It is possible to make all the pieces continuous, however, it makes the table-based approximation *different* from the underlying compact model, and is discussed in detail in Section 7. For the device models discussed here, we split the domain into a uniform grid, and inside each rectangular domain, we fit a 2D BLI.

Figure 11(a) shows the mean relative error in approximating f_e for the Penn-State Phillips (PSP) model and the Berkeley Short-channel IGFET Model (BSIM) model. In order to make the models 2-dimensional, MOSFETs have their sources tied to the bulk nodes as shown in Figure 4. For generating tables, devices are sampled in the regions $v_{gs}, v_{ds} \in [-1.5, 1.5]$V as the circuits in the later sections use $v_{dd} = 1V$. In Figure 11(a), the interpolated device functions are evaluated at 40,000 randomly selected points in $[-1, 1]$V, and the mean relative error is reported.

For both PSP, and BSIM, with about 1056 sample points per input dimension, BLI achieves near machine-precision with a mean relative error of roughly $10^{-13}\%$, or 10^{-15}. For the same number of sample points, the mean error with splines is 4-5 orders of magnitude worse. While BLI consistently outperforms splines when more than 144 sample points are used, the difference become very prominent after 288 sample points because of the underlying Chebyshev series behavior. Section 7.1 presents a detailed analysis of this in 1-dimension. Moreover, the accuracy for approximating PSP as compared with BSIM is higher with both BLI and splines. This is primarily because PSP is a more smooth model and we discuss this too in detail there.

5.4 Analyses algorithms

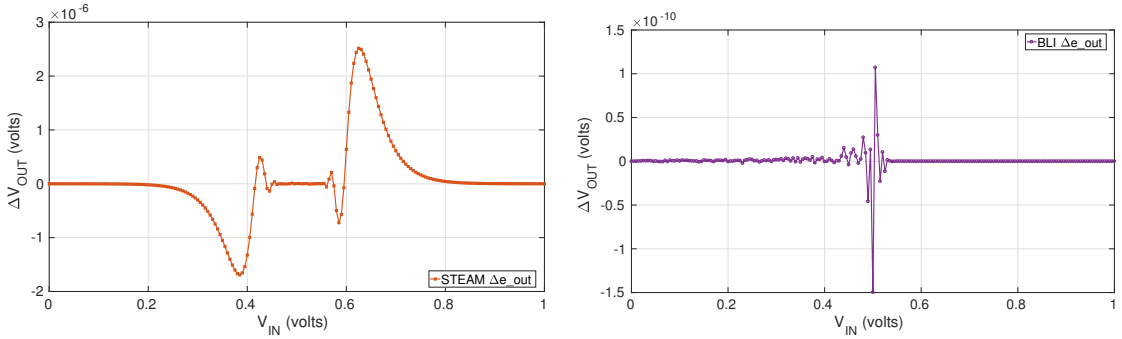


(a) 3-Stage CMOS ring-oscillator circuit

(b) Gilbert Cell mixer circuit

Figure 12: Circuits used for various analyses.

For the subsequent experiments involving analyses algorithms, we compare PSP and BSIM models with table-based approximations constructed using spline interpolants



(a) STEAM (b) BLI

Figure 13: Error in simulating a CMOS inverter for QSS using STEAM and A-LA-CARTE

CIRCUIT	Error BLI	Speedup BLI	Error STEAM	Speedup STEAM
CMOS Inverter	5.13e-12	3.58	2.23e-4	3.32
Differential Pair	1.63e-12	3.08	7.24e-9	2.84
Gilbert Cell	1.22e-12	3.46	9.98e-8	3.17

Table 2: Runtime and Error analysis for QSS with various circuits using BSIM Model.

and BLI. A 2-dimensional piecewise BLI interpolant is constructed with 32 pieces uniformly splitting the domain $[-1, 1]$V. Each piece had a local polynomial of degree 33 in both the dimensions. A 2-dimensional spline interpolant is constructed on a uniform grid of 1056×1056 points for it to have the same number of sample points as the piecewise BLI interpolant. The accuracy and speedup for these interpolants relative to the baseline models, PSP and BLI, can be seen in Figure 11(a) and Figure 11(b) respectively.

5.4.1 Quiescent Steady State (QSS) and Transient Analysis

Having compared our approach with STEAM for evaluating device functions, *i.e.*, f and q, we now analyze the overall accuracy of QSS (or DC analysis) and Transient analysis using the two approaches. We ran a QSS sweep for a CMOS inverter, and Gilbert Cell. Figure 13 shows simulation results for a CMOS inverter comparing an approximation with spline interpolants and BLI with Chebyshev sample points. As before, the error with splines is limited to $3\mu V$. With the same number of sample points, BLI achieves an error under $0.15nV$.

We also ran transient simulations for the circuits mentioned previously along with a Gilbert cell(Figure 12(b)). Figure 14 shows the simulation error and the simulation waveform for a ring oscillator circuit constructed by connecting an odd number of inverters in a circle (See Figure 12(a)). Later, in Section 5.4.2, we will analyze the same circuit using Harmonic Balance (HB) and look at the speedup and simulation

CIRCUIT	Error BLI	Speedup BLI	Error STEAM	Speedup STEAM
CMOS Inverter	6.02e-12	2.64	6.23e-4	2.48
Ring Oscillator	5.05e-12	2.91	1.48e-3	2.68
Differential Pair	3.84e-12	2.32	1.06e-8	2.21
Gilbert Cell	8.13e-12	2.75	7.90e-8	2.57

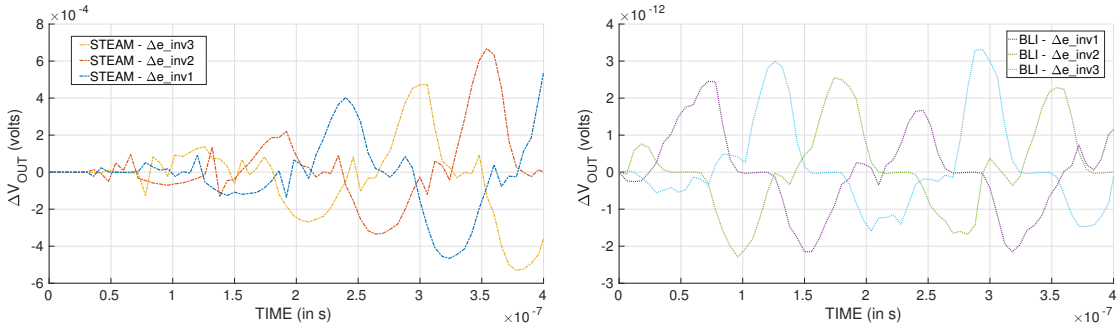
Table 3: Runtime and Error analysis for Transient Analysis with various circuits using BSIM Model.

CIRCUIT	Error BLI	Speedup BLI	Error STEAM	Speedup STEAM
CMOS Inverter	1.09e-13	8.64	1.77e-9	13.24
Differential Pair	7.93e-16	11.01	1.37e-10	10.76
Gilbert Cell	5.11e-16	13.74	2.75e-10	12.86

Table 4: Runtime and Error analysis for QSS with various circuits using PSP Model.

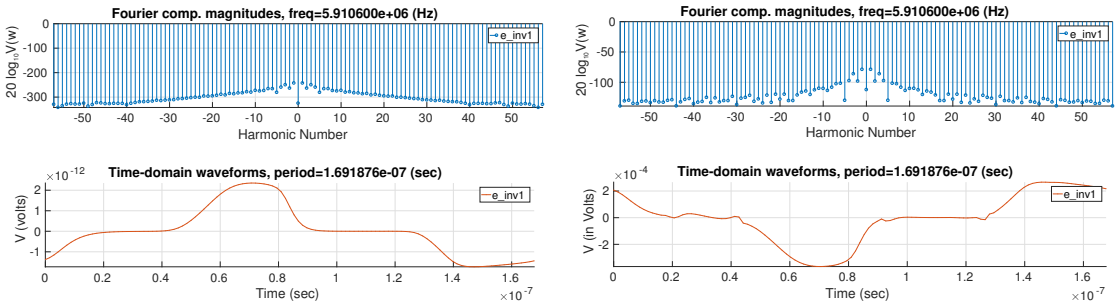
CIRCUIT	Error BLI	Speedup BLI	Error STEAM	Speedup STEAM
CMOS Inverter	4.27e-15	10.27	2.74e-10	9.92
Ring Oscillator	1.96e-14	12.13	1.49e-9	11.57
Differential Pair	6.70e-16	11.01	1.37e-10	10.76
Gilbert Cell	4.72e-13	10.19	1.10e-9	9.69

Table 5: Runtime and Error analysis for Transient Analysis with various circuits using PSP Model.



(a) STEAM (b) BLI

Figure 14: Error in Transient Simulation of a 3-stage ring oscillator STEAM and A-LA-CARTE



(a) BLI (b) STEAM

Figure 15: Time-Frequency domain error in Harmonic balance for a 3-stage ring oscillator circuit.

error in a more accuracy sensitive application. The results with PSP and BSIM models for various analyses and circuits are summarized in Tables 2, 3, 4, and 5. Figure 14 shows the error in transient simulation of a 3-stage ring oscillator (see Figure 12(a) for the circuit) using STEAM and A-LA-CARTE. Here too, A-LA-CARTE improves the simulation accuracy by 8 orders of magnitude.

5.4.2 RF design - Harmonic Balance

While previous attempts at table-based device modeling have made steady progress towards accuracy improvements, none have so far reached a level that is adequate for sensitive applications, like RF design. The achievement of near machine precision in the previous section (Section 5.3) motivates the use of our approach for table-based device modeling in these applications. Some of the methods used for analyzing analog circuits that have periodic behavior include Periodic Steady State (PSS) (or Shooting) and HB. In this section, we compare the results from the approach used in [3] and our approach. Accuracy sensitive applications, like RF design, highlight the important differences in accuracy between the two methods. This is especially true for a method like HB that implicitly depends on the accuracy of higher order derivatives.

HB is a mixed time-frequency domain analysis that is used for analyzing oscillatory circuits. It can be applied to both autonomous systems, like oscillators, and systems driven by periodic signals like amplifiers. We applied both the approaches to accelerate HB for computationally expensive simulations. For the former case, *i.e.*, analysis of oscillators, HB computes both the oscillation frequency and the amplitudes of various harmonics corresponding to the oscillation frequency. This is shown in Figure 15. Here, both BLI and spline interpolants were constructed by using 1056 sample points for each of the two input dimensions. It can be seen that BLI has an error of $\sim 10^{-12}$V in a peak voltage of $1V$, whereas spline interpolant has an error of $\sim 10^{-4}$V.

6 Sparse measurements

So far, we have been relying only on the data that we have sampled from a device model in order to construct a polynomial approximation. The prior knowledge of the underlying smoothness of device models has been exploited only to the extent of rapid drop in Chebyshev series coefficients. As discussed in Section 5, this gives a very accurate approximation with Barycentric-Lagrange Interpolation (BLI) when using Chebyshev sample points.

An alternate approach can be used to make use of the underlying model smoothness. Having obtained a few sample values from the device functions, we can use the smoothness assumption to 'fill' the gaps that have not been sampled yet. Such an approach can have a lot of practical utility in situations where evaluating the device functions is computationally more expensive than the 'filling' process mentioned above. Moreover, this could also be used in situations where measurement data from physical devices is used for constructing models as individual measurements could be cumbersome and limited.

6.1 Compressibility of Device data

In order to assess the feasibility of such methods, it is essential to estimate the amount of redundancy in the data that we are measuring to construct our polynomial approximations. Low rank decomposition, and basis transformations are some of the most commonly used methods for this purpose and we explore these in the context of device model data to analyze how much data is *needed* for the accuracy levels that we have previously mentioned. At the same time, for a given amount of data, we can understand how much accuracy can be achieved by artificially generating the data based on prior knowledge.

6.1.1 Singular Value Decomposition

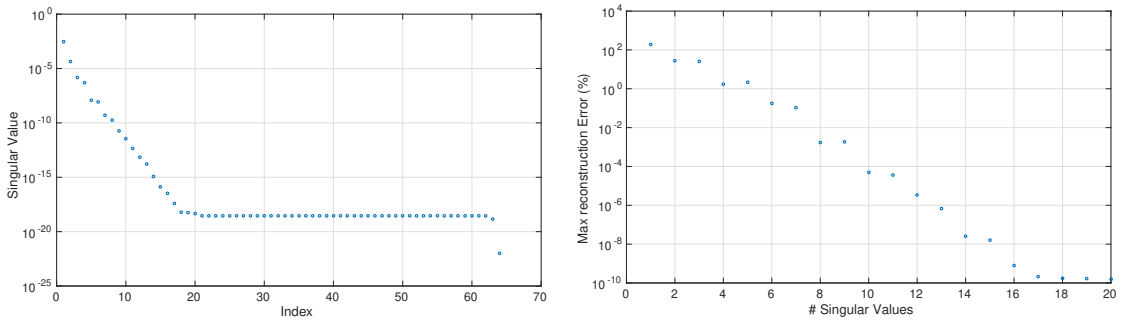
Singular Value Decomposition (SVD) serves as a very useful tool in assessing matrices. The idea behind SVD is to express a matrix $M_{m\times n}$[10] as a sum of rank-1 matrices, *i.e.*,

$$M = \sum_{1\leq i\leq min(m,n)} \sigma_i u_i {v_i}^T, \tag{22}$$

where u_i and v_i are unit-norm column vectors sized $m \times 1$, and $n \times 1$ respectively. The *weights* of the rank-1 matrices $u_i v_i^T$ are the singular values.

Figure 16 looks at the singular values of a matrix of 64 I_{ds} values for an NMOS modelled with BSIM as depicted earlier in Figure 4. In order to obtain these values, v_{db}, and v_{gb} are varied on an 8×8 rectangular grid of points. Figure 16(a) shows that successive singular values for this matrix drop very rapidly, levelling off to numerical precision error after 18 singular values. The impact of this on compressibility of the matrix can be seen when only the k-highest singular values, and the corresponding

[10] $M_{m\times n}$ is a matrix with m rows and n columns.



(a) SVD for a 2D slice.

(b) Reconstruction with k-highest singular values.

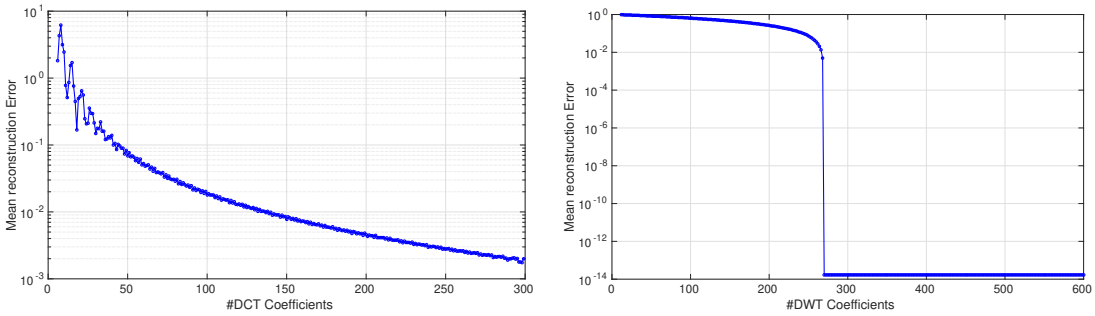
Figure 16: SVD on a 2D slice of I_{ds} data for a NMOS using BSIM.

basis vectors u_k, and v_k are used to approximate the matrix. We get an approximation of M, called $\hat{M}$, where,

$$\hat{M} = \sum_{1 \leq i \leq k} \sigma_i u_i v_i^T. \tag{23}$$

The corresponding error in this approximation can be measured as $\max_{i,j} \left(\frac{\hat{M}(i,j) - M(i,j)}{M(i,j)} \right)$[11], or the highest value of point-wise relative error. This is shown in Figure 16(b), where, one can see that with only 10 singular values, the maximum relative error drops to 0.0001% or 10^{-6}. After 17 singular values, this flattens out at 10^{-12} of the corresponding values in M.

6.1.2 Basis Transformations



(a) Discrete Cosine Transform (DCT)

(b) Discrete Wavelet Transform (DWT)

Figure 17: Reconstruction error for a 1-dimensional slice of f using the highest k transform coefficients as k is varied.

SVD is fairly informative of the structure and redundancy in the data available. However, it still does not explicitly make use of the fact that the model data must be smooth. This can easily be incorporated by representing the device model data

[11]Matrix division is done element by element.

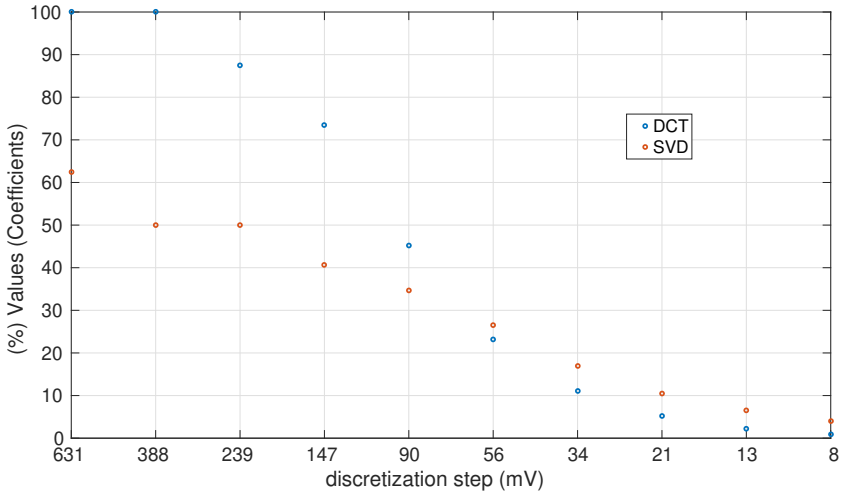


Figure 18: Comparing partial reconstruction using SVD and DCT.

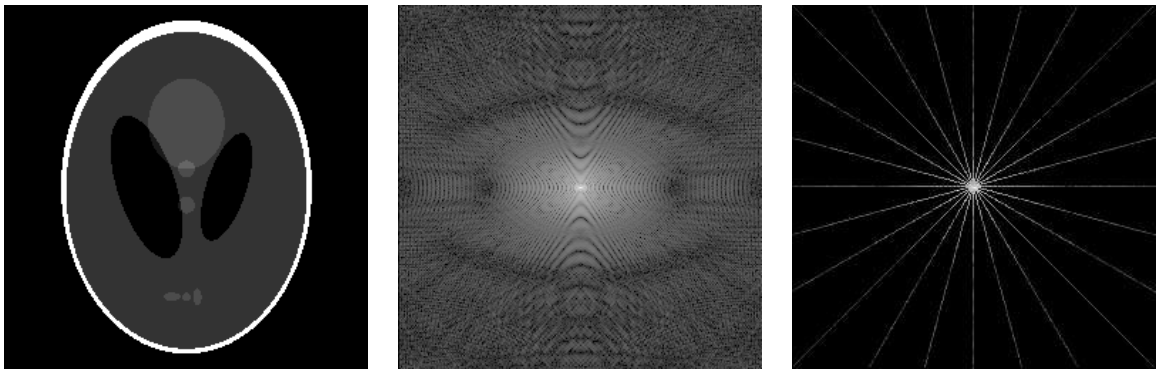
in a basis which has smooth elements. DCT basis elements, as well several wavelet families satisfy these requirements very well. Reconstruction error using a partial set of basis coefficients is shown in Figure 17 for a total of 600 values for a model with a single input. One of the reasons for the relatively poor performance of DCT is that DCT inherently enforces boundary conditions. In that respect, it is similar to Discrete Fourier Transform (DFT), which enforces a periodic boundary condition. As in our case, if the signal or function being approximated with truncated DCT coefficients does not meet the boundary conditions, a large amount of error is seen near the boundaries.

Figure 18 exemplifies the essential difference between a low-rank approximation (using SVD) and a smooth approximation of the device model data. For a given level of accuracy (in this case, 3 decimal digits), Figure 18 shows the percentage of coefficients (singular values or DCT weights) needed to achieve this accuracy. Additionally, the matrix size is varied here, and is inversely proportional to the discretization step on the x-axis. For a discretization step of 631mV, the matrix would be 4×4, and for 8mV, it would be 250×250. For small matrices of device model data, SVD seems to require relatively fewer entries, however, as the total amount of data grows, basis transforms like DCT see greater improvement in their ability to compress the device model data.

6.2 Recreating a picture pixel-by-pixel

The observations in Section 6.1.2 tell us that the device model data is *sparse* in some common bases, such as DCT and some families of wavelets. From a numerical standpoint, many of the smaller basis coefficients do not contribute to the actual value of the function, making it sparse. Compressed Sensing (CS) [21, 22, 23, 24] was built around reconstruction of signals that could be sparsely represented in some basis using fewer measurements than required by Shannon-Nyquist sampling theorem.

Figure 19 demonstrates one of the first applications of CS. In Figure 19(a), a 256×256 phantom image is shown. Its Fourier domain coefficients can be seen in Figure 19(b). Only the absolute value of the coefficients is shown on a logarithmic

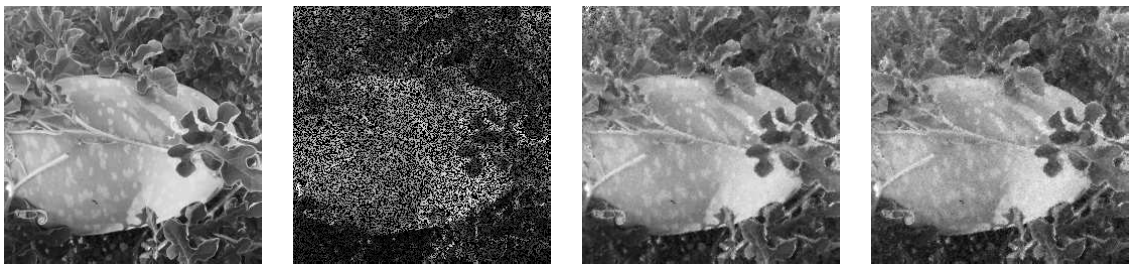


(a) Original image (b) True Fourier Coefficients (c) Observed Fourier Coefficients

Figure 19: Working example of CS

scale for better visibility. Note that the contrast on the logarithmic scale shows that the coefficients are indeed very sparse. We are able to sample these Fourier coefficients on any radial line in the Fourier space, and Figure 19(c) shows one such measurement. In CS, the task is to reconstruct the original image in Figure 19(a), such that the Fourier domain representation of the reconstruction matches the measured values at the points where measurement is available (Figure 19(c)).

Interestingly, subjecting the reconstruction to have the least *total variance* in the Fourier domain coefficients leads to an exact match with the original image. In this case, we wish to *reconstruct* an image I, while we can *measure* its Fourier coefficients. Such situations naturally arise in Magnetic Resonance Imaging (MRI) and Computational Tomography (CT), where measurements indeed correspond to Fourier representations. On the other hand, in our framework (Spline-based Tables for Efficient and Accurate device Models (STEAM)), we can measure the *image* or a function at any point, and the goal is to reconstruct the same image (function). We do know, from prior experience, that the image would be sparse when represented in the DCT basis, or the wavelet basis.



(a) Original Image (b) Random samples of pixel values (c) Reconstruction with TV Minimization (d) Reconstruction with $L1$ minimization

Figure 20: A variant of the phantom experiment that leads to device evaluation

This analogy is depicted in Figure 20. In 20(a), we have the true image that we are trying the measure. In order to do that, we only measure the image at a sparse set

of points, such that the number of measurements we make is significantly lesser than the total number of pixels that we would like to reconstruct. Thereafter, we can reconstruct the image by enforcing the sparsity of the DCT coefficients, while simultaneously forcing the reconstruction to match the measured values. Reconstruction obtained by enforcing sparsity with a TV minimization (as in the phantom experiment), and $L1$ norm minimization (conventional CS) are shown in Figures 20(c), and 20(d) respectively. For these, and all future optimization experiments presented here, L1-magic was used, which is a publicly available software package.

Let us formulate both the approaches mentioned above for a 1-dimensional signal. An image can be converted to a 1D signal by concatenating all the columns into a single one. Let us call the baseline signal or function, I. In the former approach, measurements of I in a basis $\mathcal{F}$ are available, *i.e.*,

$$T_I = M\mathcal{F}I. \tag{24}$$

Here, M is the measurement matrix. In Figure 19, $\mathcal{F}$ is the Fourier transform matrix, whereas the measurement matrix M selects values along a radial line. Let F_I correspond to the Fourier-domain coefficients for I. Then, F_I can be solved from the under-determined system in Equation 24 by solving

$$\arg\min_{\|x\|_1} \; s.t.\; T_I = Mx. \tag{25}$$

The actual image I is obtained by inverting the Fourier transform, *i.e.*, $I_r = \mathcal{F}^{-1}x_{sol}$, where, I_r is the reconstructed image, and x_{sol} is the solution to Equation 25. The later method, one we might call *image completion*[12], involves a situation where the some of the pixel values are known. In addition to this, we also assume that the image has a sparse representation in some basis $\hat{\mathcal{F}}$. Again, let M be a measurement matrix. Since we are sampling a few discrete points in the original image, M depends on the points being sampled, and is known. Now,

$$\begin{aligned} MI &= M\hat{\mathcal{F}}^{-1}x, \\ i.e.\; y &= M\hat{\mathcal{F}}^{-1}x. \end{aligned} \tag{26}$$

Here, y refers to a measurement of a few pixel values of the original image I, x refers to the coefficients corresponding to the image I in the basis $\hat{\mathcal{F}}$. The coefficient vector, x, can be obtained by solving the following optimization problem:

$$\arg\min_{\|x\|_1} \; s.t.\; y = M\hat{\mathcal{F}}^{-1}x. \tag{27}$$

The reasoning behind the working of Equation 27 lies in the fact that under some circumstances, the solution x_{sol} to Equation 27 not only has the smallest L1-norm, but is also the most sparse vector to satisfy the constraint $y = M\hat{\mathcal{F}}^{-1}x_{sol}$, *i.e.*, x_{sol} is also the solution to

$$\arg\min_{\|x\|_0} \; s.t.\; y = M\hat{\mathcal{F}}^{-1}x. \tag{28}$$

[12]This refers to matrix completion, which is essentially the same problem without the prior knowledge that the matrix represents a smooth function or an image.

In [21, 22, 23], it was shown that the solution to 28 is same as the solution to 27, if the orthonormal transform $\hat{\mathcal{F}}^{-1}$ satisfies:

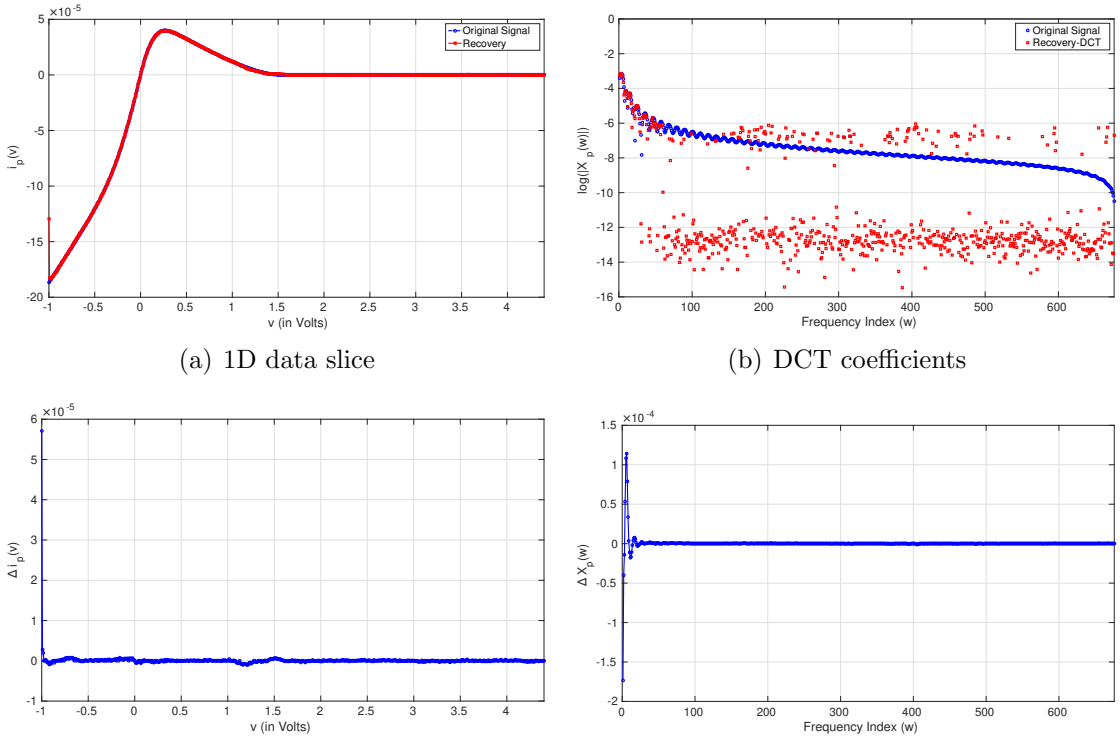
$$\mu(\hat{\mathcal{F}}) = \max_{0 \leq i,j \leq N} \hat{\mathcal{F}}(i,j) > \frac{N}{s}, \tag{29}$$

where $\mathcal{F}_{i,j}$ corresponds to the (i,j) entry in $\mathcal{F}$ represented as a matrix, and the vector x is s-sparse. $\mu(\hat{\mathcal{F}})$ is the 'coherence' of the transform $\hat{\mathcal{F}}$ [24] and is a scalar value. Simply put, for a given transform $\hat{\mathcal{F}}$, if S_x denotes the set of coefficient vectors that match the partial measurements specified by A, then

$$\arg\min_{x \in S_x} \|x\|_0 = \arg\min_{x \in S_x} \|x\|_1. \tag{30}$$

In [24], it was shown that if I has an $s - sparse$ representation in $\hat{\mathcal{F}}$, then, the solution to Equation 30 uniquely identifies I.

6.3 Sparse sampling and reconstruction of device data



(a) 1D data slice

(b) DCT coefficients

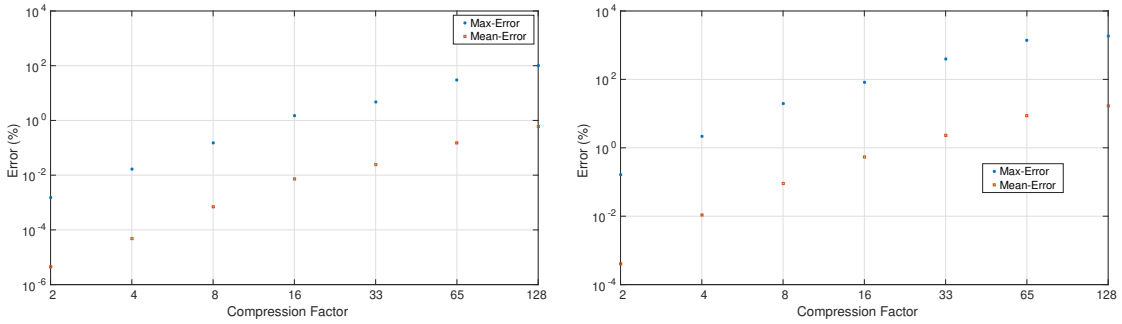
(c) Recovery error

(d) DCT coefficients

Figure 21: Recovery of BSIM I_{ds} with a compression factor of 8.

The recreation from sparse samples in Section 6.2 can be applied to the device data too. Figure 17 shows the results for this for a 1-dimensional data slice. The original signal and its reconstruction are shown in Figure 21(a). For reconstruction, first a few points are randomly sampled. Then, over possible DCT coefficient values that match

measured data after application of inverse DCT transform, one with the minimum $L1$-Norm is chosen. Figures 21(d) and 21(c) show the error in the reconstructed DCT coefficients and waveform respectively. Again because of the periodic nature of DCT, the error in the reconstruction is relatively high at the left boundary.



(a) 2mV step (b) 8mV step

Figure 22: Error in reconstructing I_{ds} matrix obtained by sampling $f(v_{ds}, v_{gs})$ with different discretization steps as compression factor is varied.

Figure 22 extends the previous experiment to 2 dimensions. The error at the boundary is handled by choosing a domain that extends beyond the desired boundary by a small amount and ignoring the error in this extension strip. 2 different discretization steps are chosen, and for each, compression factor represents the fraction of values that are randomly sampled for reconstructing data over the entire grid. Interestingly, for the same compression factor, we get better accuracy on a finer grid. If for an application, $\sim 1\%$ maximum error is tolerable, then with this method, $16\times$ fewer model evaluations are needed to obtain the data corresponding to a $2mV$ discretization step. The key result here is that the MODel SPECification (MODSPEC) functions, that completely describe the behavior of a device can be reconstructed with a very high accuracy by a few samples in the input dimensions.

7 Model diagnosis

As we have mentioned on several earlier occasions, compact models are prone to several mathematical problems like discontinuities arising from hand-coded model equations. These discontinuities degrade the performance of the model in simulation by taking more Newton Raphson (NR) iterations to converge. More importantly, in Radio Frequency (RF) design, derivative discontinuities result in incorrect harmonics.

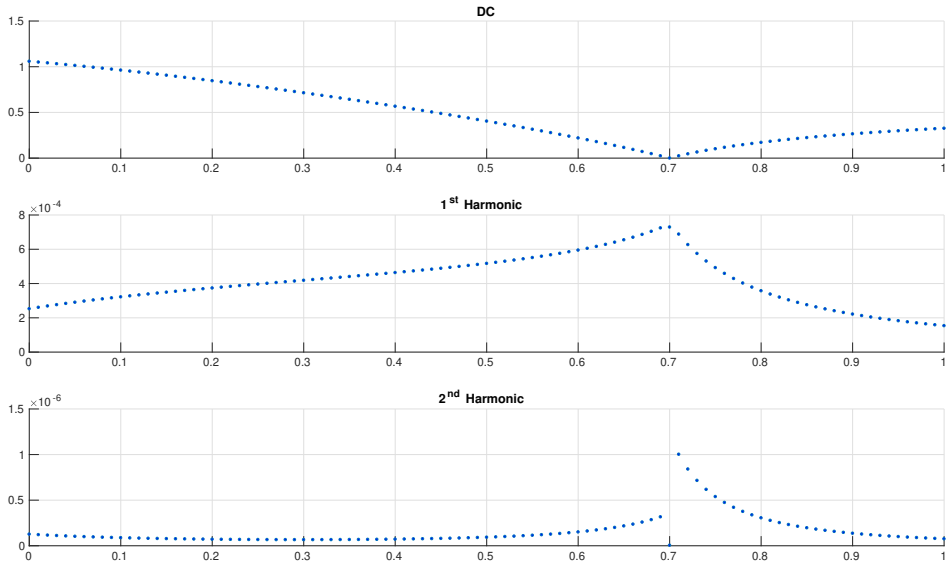


Figure 23: HB on a differential pair.

Figure 23 analyzes the outcome of HB on a differential pair (circuit shown earlier in Figure 7) with the Shichman-Hodges (SH) model. In this experiment, the input bias voltage, V_{in}, is varied from 0 to V_{dd}, and for each bias value, HB is executed on the circuit to get the transfer function from the small-signal input v_{in} to the output v_{out}. Both the 1^{st} and 2^{nd} harmonics obtained by HB while sweeping v_{in} are noteworthy. The second harmonic in the bottom panel abruptly drops to 0 for $v_{in} = 0.7$, which is unrealistic. SH is a fairly crude model of a transistor, and lacks continuous 2^{nd} derivatives. The idea behind this exercise is not to emphasize the choice of a model for RF design. We are simply trying to demonstrate how derivative discontinuities in a model seep into physical quantities that are of interest in real designs.

In Section 5, it was mentioned that discontinuities in the models force us to split the domain and fit different polynomials to each domain. This was required because there, the goal was to faithfully replicate the output or response of a given compact model. Having looked at the affect of these discontinuities in the simulation results, we will now take another look at them in the context of table-based modeling. We will see that the observation of *polynomial-ringing* in Section 5.2 can be used to diagnose and pin-point model discontinuities. Moreover, we will see that if the discontinuities were not present, device models could be approximated with A LAgrange-interpolant with Chebyshev-samples for Accurately Representing TablE-models (A-LA-CARTE) using very very few points at near machine precision.

7.1 An illustrative 1D Example

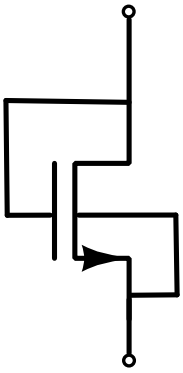
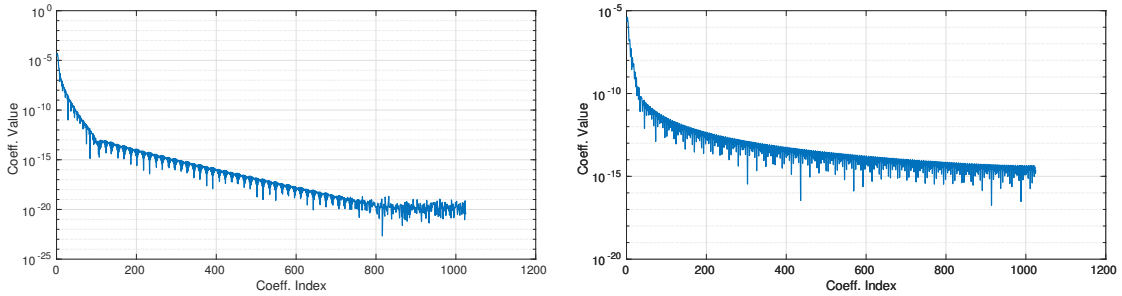


Figure 24: Schematic of a diode connected MOSFET

Consider a diode connected MOSFET as shown in Figure 24, modeled with Berkeley Short-channel IGFET Model (BSIM) and Penn-State Phillips (PSP) models with only one input, called v_c. For simplicity, we have ignored terminal resistances from the model, making it a 1-port device. It can be completely characterized by providing, for any terminal voltage v_c, a DC current term i_c and a dynamic term q_c. A table-based model for the diode-connected MOSFET must reconstruct these two functions, $i_c(v_c)$, and $q_c(v_c)$.

Figure 25(a) shows the Chebyshev series coefficients for the input current i_c obtained from the PSP model. **The view so far has been:** "*After $\sim$ 800 indices, the coefficients are not numerically significant to contribute to any error in the table-based model. Therefore, if we were to construct an interpolant with more than 800 Chebyshev sample points, it would reconstruct the model to near-machine precision.*" We can compare these Chebyshev series coefficients with those for continuous and discontinuous functions (or smooth and non-smooth functions) for some visual insight.



(a) PSP (b) BSIM

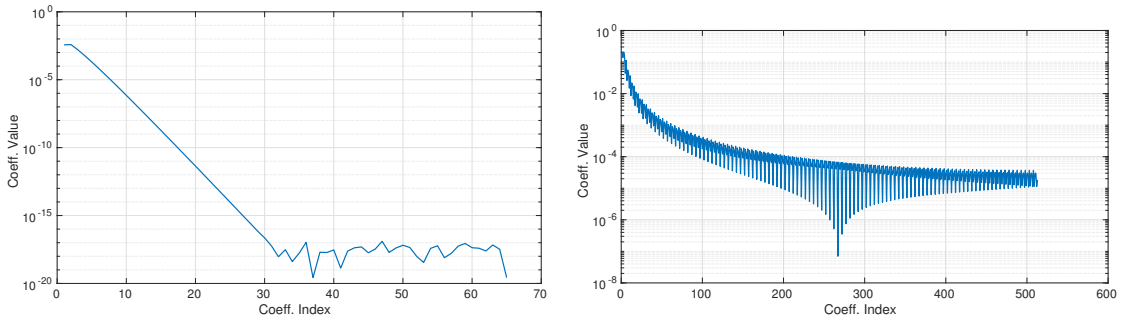
Figure 25: Showing the first 1024 Chebyshev series coefficients for i_c of a diode-connected MOSFET model.

Figure 26 shows the typical shape of Chebyshev series coefficients for smooth (Figure 26(a), where the function has continuous value, first, and higher order derivatives), as well as non-smooth functions (Figure 26(b), where there is discontinuity in first derivative). There are two important observations to be made from the plots above.

1. Chebyshev series coefficients for i_c from diode connected BSIM model bear strong resemblance to the coefficients for a *non-smooth* function in Figure 26(b), and

2. Coefficients for i_c from the PSP model drop rapidly initially (by $\sim$ 10 orders of magnitude over the first 100 coefficients), but later on, show very slow decay and some ringing.

There could be two explanations for the observations above:



(a) Smooth function (b) Non-smooth function

Figure 26: Typical Chebyshev series coefficients for smooth *vs.* non-smooth functions.

1. Device model functions are inherently more complex that the simple examples in Figure 26, and this is makes the use of high-order polynomial representations necessary, or

2. Both PSP and BSIM models are discontinuous to some extent, and if we could find these discontinuities, and fit interpolants to individual pieces, the Chebyshev series coefficients for each of these pieces would resemble 26(a).

7.2 Notion of accuracy, domain splitting

Method	Pieces	Total Coefficients	Measured Error
Single Polynomial	1	257	5.24e-6
	1	513	1.68e-8
	1	1025	1.136-12
	1	2049	5.98e-13
Uniform Pieces	16	272	9.12e-7
	32	544	2.42e-9
	32	1040	4.90e-13
	32	2080	4.89e-13
Hand-picked Split	8	70	4.92e-13

Table 6: Required Polynomial Coefficients (or table size) for reconstructing PSP - f.

If one looks at the segment of Table 6 that refers to *single piece* as the method, one can notice that the mean point-wise relative error does indeed drop significantly when the number of sample points is dropped from 513 to 1025. This is consistent with the fact that after $\sim$ 800 Chebyshev series coefficients, remaining values are numerically insignificant. Similar observation can also be made for the BSIM model in Table 7,

Method	Pieces	Total Coefficients	Measured Error
Single Polynomial	1	257	3.81e-4
	1	513	1.15e-4
	1	1025	2.66e-5
	1	2049	4.68e-6
Uniformly Spaced	16	272	6.59e-5
	32	544	7.23e-6
	32	1040	1.54e-6
	32	2080	1.17e-7
Hand-picked Split	8	40	3.35e-14

Table 7: Required Polynomial Coefficients (or table size) for reconstructing BSIM-f.

where both the Chebyshev series coefficients and the interpolation error do not become numerically insignificant.

Here, it is very important to note that this measure or notion of *accuracy* is based entirely on how *different* the table-based model is from the original model. Let us now see to what extent can Piecewise Barycentric-Lagrange Interpolation (BLI) cut down on the memory requirement for this case and simultaneously increase the interpolation accuracy.

Table 7 shows a summary of reconstruction results for the BSIM model. For this experiment, the model was sampled in $[-1, 1]$V using Chebyshev points described in Section 2.5. After constructing the interpolant, it was evaluated at 10000 points randomly scattered in $[-1, 1]$V and the mean relative error was reported for these experiments. For the first 4 entries in Table 7, the total number of sample points, and therefore the degree of the interpolating polynomial is increased from 257 to 2049. Then, instead of using a single high order polynomial, we use lower order polynomials for different domains.

For example, **pieces**=8, and **Total Coefficients**=40 with uniform pieces would correspond to a polynomials of degree 5 fit to domains $(-1, -0.75)$, $(-0.75, -0.5)$, $\ldots$, $(0.75, 1)$ independently. For each of these domains, we first generate 5 Chebyshev points, *i.e.*, $(-1, -\sqrt{2}, 0, \sqrt{2}, 1)$. These points then need to be scaled to the appropriate domain, for example, for the domain $(0.5, 0.75)$, these points would approximately be $(0.500, 0.537, 0.624, 0.713, 0.750)$. The function value, in this case i_c, is evaluated at these points, and a polynomial is constructed to interpolate these values if later on during simulation, a voltage v_c in the range $(0.5, 0.75)$ is queried.

7.3 How and Why Piecewise BLI "helps"

Figure 28(a) shows the reconstruction error in the current i_c for a polynomial of order 1025. The error has a marked peak near $x(v_v) \sim -0.3V$, which prompts further

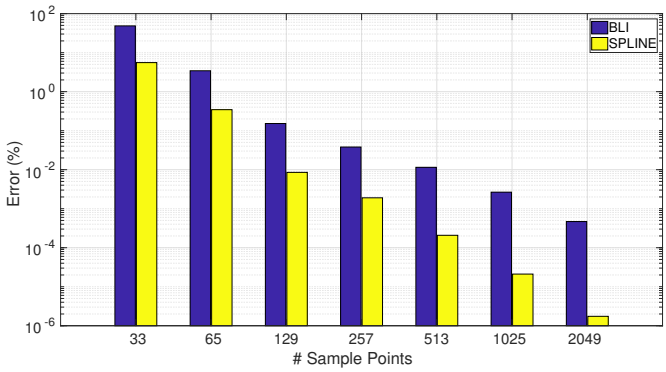
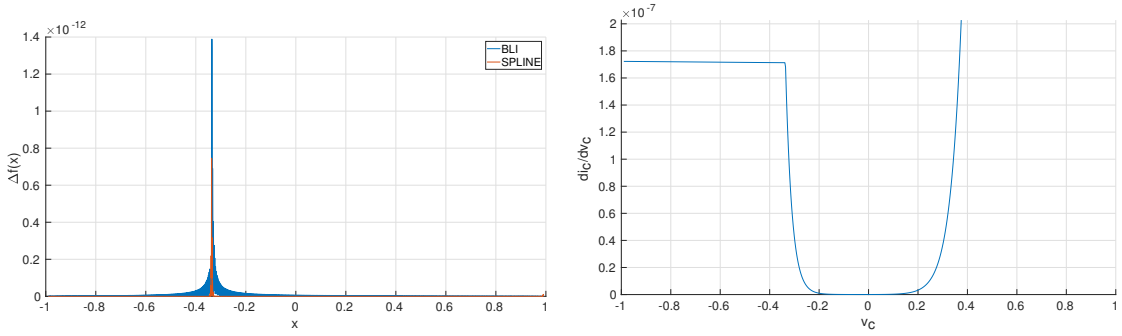


Figure 27: Reconstruction error in total current, i_c, as v_c is varied

investigation. The ringing that we see here is similar to the one observed earlier in Section 5.2.

Figure 28(b) shows the derivative of the current i_c with respect to the voltage v_c using the BSIM model. This is obtained from the original model by combining the derivatives $\frac{d}{v_{gs}} i_{gs}$, $\frac{d}{v_{gs}} i_{ds}$, $\frac{d}{v_{ds}} i_{gs}$, and $\frac{d}{v_{ds}} i_{ds}$. The first derivative thus obtained is not differentiable at the same point where we see the peak in interpolation error.



(a) Error in Total Current i_c

(b) Derivative of Total Current

Figure 28: Analyzing the reported inaccuracy in BSIM's reconstruction with BLI.

Breaking the polynomial interpolant into smaller lower-order polynomials by finding the appropriate breakpoints removes these problems as seen in Section 5.2. The last entries in both Tables 6, and 7 show that it is indeed possible to cut down the cost of memory dramatically for representing these models with an appropriate choice of break points. While these problems can largely be alleviated by switching to a model that is more mathematically sound, like PSP, errors in higher order derivatives are inevitable in such complex models.

For both BSIM and PSP the total number of sample points required to get to near machine-precision is far fewer than what can be achieved with ad-hoc measures like uniform splitting. For BSIM, just 40 sample points suffice in 1 dimension for achieving relative error close to 1e-14, and for PSP, 70 points are required for the same. Besides, we've seen that fitting a single piece BLI with Chebyshev sample points, and looking at the Chebyshev series can first help determine if the model has

discontinuities. Further, looking at the reconstruction error can help us pinpoint the location of such a discontinuity.

8 Open questions

In this thesis, we explored a few methods and applications for table-based device modeling. Along the way, we were pressed by both challenging questions and opportunities that could not be fully covered here. Some of these are discussed below.

Precise speedup for higher dimensions

There is some preliminary evidence suggesting that in 3-dimensions, table-based models, especially derived using A LAgrange-interpolant with Chebyshev-samples for Accurately Representing TablE-models (A-LA-CARTE), would provide sufficient accuracy and speedup to be practically usable. Most of our implementation has been in MATLAB and octave so far. The interpretive nature of these languages makes numerical speedup numbers both unreliable and impractical as most efficient simulators rely on compiled languages. An implementation in C/C++ would be necessary for a reliable estimate of the speedup that can be achieved.

Compressed sensing with Chebyshev series

The Compressed Sensing (CS) experiments currently rely on a uniform discretization of the domain. A sparse set of samples are obtained from this uniform grid. Preliminary experiments show that if samples are obtained from a Chebyshev grid, then, CS can directly reconstruct a truncated Chebyshev series. We have seen that for the same compression factor, this reconstruction is far more accurate than one on a uniform grid.

Table-based models from measurements

The ultimate goal of table-based modeling would be to construct models directly from device measurements. Our approach is limited by internal unknowns and the model structure which cannot be extracted trivially from measurements at the device terminals. A more wholistic table-based modeling approach which can estimate not only device functions, but the structure of internal nodes for producing a model would be more useful in practice.

References

[1] G. Gildenblat, X. Li, W. Wu, H. Wang, A. Jha, R. Van Langevelde, G. D. J. Smit, A. J. Scholten, and D. B. M. Klaassen. Psp: An advanced surface-potential-based mosfet model for circuit simulation. *IEEE Transactions on Electron Devices*, 53(9):1979–1993, 2006.

[2] Y. Cheng and C. Hu. *MOSFET modeling & BSIM3 users guide*. Springer Science & Business Media, 1999.

[3] A. Gupta, T. Wang, A. G. Mahmutoglu, and J. S. Roychowdhury. STEAM: Spline-based tables for efficient and accurate device modelling. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 463–468, Jan 2017.

[4] C. De Boor. *A practical guide to splines*, volume 27.

[5] J. Berrut and L. N. Trefethen. Barycentric Lagrange interpolation. *SIAM review*, 46(3):501–517, 2004.

[6] A. Gil, J. Segura, and N. Temme. *Numerical Methods for Special Functions*. Society for Industrial and Applied Mathematics, 2007.

[7] R. B. Platte and L. N. Trefethen. Chebfun: a new kind of numerical computing. *Progress in industrial mathematics at ECMI 2008*, pages 69–87, 2010.

[8] W. M. Coughran Jr, E. Grosse, and D. J. Rose. CAzM: A circuit analyzer with macromodeling. 30(9):1207–1213, 1983.

[9] W. R. Richards. CAzM 5.0 - A Robust, Table-Based Analog Circuit Simulator. In *The 24th Southeastern Symposium on System Theory and The 3rd Annual Symposium on Communications, Signal Processing Expert Systems, and ASIC VLSI Design*, pages 459–462, Mar 1992.

[10] T. Shima and H. Tamada. Table look-up MOSFET modeling system using a 2-D device simulator and monotonic piecewise cubic interpolation. 2(2):121–126, 1983.

[11] J. A. Barby, J. Vlach, and K. Singhal. Polynomial splines for mosfet model approximation. 7(5):557–566, 1988.

[12] V. Bourenkov, K. G. McCarthy, and A. Mathewson. MOS table models for circuit simulation. 24(3):352–362, 2005.

[13] R. Kanj, T. Li, R. Joshi, K. Agarwal, A. Sadigh, D. Winston, and S. Nassif. Accelerated statistical simulation via on-demand hermite spline interpolations. pages 353–360. IEEE, 2011.

[14] X. Li, F. Yang, D. Wu, Z. Zhou, and X. Zeng. MOS table models for fast and accurate simulation of analog and mixed-signal circuits using efficient oscillation-diminishing interpolations. 34(9):1481–1494, 2015.

[15] A. R. Rofougaran, B. Furman, and A. A. Abidi. Accurate analog modeling of short channel fets based on table lookup. pages 13–1. IEEE, 1988.

[16] D. Amsallem and J. S. Roychowdhury. ModSpec: An open, flexible specification framework for multi-domain device modelling. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 367–374. IEEE, 2011.

[17] A. Townsend and L. N. Trefethen. An extension of chebfun to two dimensions. *SIAM Journal on Scientific Computing*, 35(6):C495–C518, 2013.

[18] N. J. Higham. The numerical stability of barycentric lagrange interpolation. *IMA Journal of Numerical Analysis*, 24(4):547–556, 2004.

[19] V. Barthelmann, E. Novak, and K. Ritter. High dimensional polynomial interpolation on sparse grids. *Advances in Computational Mathematics*, 12(4):273–288, 2000.

[20] T. Wang, A. V. Karthik, B. Wu, and J. S. Roychowdhury. MAPP: A platform for prototyping algorithms and models quickly and easily. In *IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO)*, pages 1–3. IEEE, 2015.

[21] E. J. Candès, Y. C. Eldar, D. Needell, and P. Randall. Compressed sensing with coherent and redundant dictionaries. *Applied and Computational Harmonic Analysis*, 31(1):59–73, 2011.

[22] E. J. Candès, M. Rudelson, T. Tao, and R. Vershynin. Error correction via linear programming. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 668–681. IEEE, 2005.

[23] E. J. Candès and T. Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

[24] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006.

[25] Emmanuel Candes and Justin Romberg. l1-magic: Recovery of sparse signals via convex programming. *URL: www.acm.caltech.edu/l1magic/downloads/l1magic. pdf*, 4:14, 2005.

9 Appendix: Code

The software used for various parts of this project has been compiled in the form of a package titled STEAM. It is publicly available at https://github.com/architgupta93/STEAM. STEAM consists of 6 main sub-parts:

1. **Berkeley Modeling and Algorithm Prototyping Platform (MAPP)**: An open-source circuit simulator in MATLAB [20],
2. **Piecewise-Polynomial Interpolation (PPI)**: A polynomial interpolation package that implements all the interpolants mentioned in the report (splines, Chebyshev polynomials, Chebyshev series interpolant, Discrete Cosine Transform (DCT) based interpolants, *etc.*),
3. **L1-Magic**: Software provided open-source by Justin Romberg for Compressed Sensing (CS) [25],
4. **Device-Models**: A set of compact models (Berkeley Short-channel IGFET Model (BSIM), Penn-State Phillips (PSP) for example), translated into the MODel SPECification (MODSPEC) modeling API using Verilog-A Parser and Preprocessor (VAPP),
5. **Core framework**: The framework for translating any device model expressed in the the MODSPEC modeling API into a table-based approximation using either Splines, or Chebyshev interpolants, and
6. **Examples**: These are distributed into two locations. A directory called '*examples*' under STEAM provides examples that demonstrate the use of the package, and another set of examples under '*ThesisExtras*' which reproduce the results presented here.

The main aspects of software that we implemented are described here. For additional information on the usage/extension of the package, we encourage you to read the package documentation.

9.1 PPI

The PPI package can be used to approximate functions with different polynomial interpolants that use a common API. Interpolants used in this report can be found under `bli`, and `splines` in the package directory. PPI package has some additional features which were not used here. The `examples` directory in this package has a list of examples to demonstrate its functionality independently of the parent package.

Files `illustrative_example.m` and `univariate/fx_bli.m` are good starting points. The former script demonstrates the use of splines and the latter compares Barycentric-Lagrange Interpolation (BLI) and splines for a given function.

9.2 Report results

Here, we walk the reader through the supplied example code and how it can be used to reproduce the main results presented in this report. Prior to running any of the examples mentioned here, MATLAB PATH must be set up appropriately. Scripts are provided for the this and users should refer to the supplied documentation to access them.

9.2.1 Chebyshev polynomials and Chebyshev points

From `ThesisExtras`, run `chebyshev_polynomials.m` to generate Chebyshev polynomials of the first kind. The supplied script allows to to vary the degree of the Chebyshev polynomial being drawn. Similarly, Chebyshev Points can be visualized using the example code `chebyshev_points.m` in the same location. This also allows you to visualize a Chebyshev points of different orders simultaneously. These orders can be specified in the script to look at Chebyshev points of different orders. We demonstrate orders 8 and 16 to show the embedding of Chebyshev points.

9.2.2 Passive extrapolation

Within PPI, passive extrapolation can be seen. With `examples/univariate/fx_extrapolation.m` in PPI package, reader can compare Not-A-Knot spline and passive spline. The code for generating a passive spline (in general a spline interpolant with a given boundary slope can be found in `splines/PassiveSpline1D.m`). This reuses bulk of the code for generating a spine interpolant from a given set of knots and associated functions values in `splines/SplineInterpolant.m`.

9.2.3 STEAM: Device evaluation

With `ThesisExtras/device_evaluation_speedup.m`, one can evaluate the speedup and accuracy of STEAM for approximating any given MODSPEC model. It also serves as a guide for using STEAM with your own models. Currently, only 2-dimensional models (*i.e.* models with 2 inputs) can be translated with STEAM. STEAM takes in a MODSPEC model and a set of interpolation parameters (see `STEAMArgs.m` in utils for an overview) to create a polynomial approximation. This will generate the accuracy and speedup results for a single device model.

9.2.4 STEAM: Circuit analyses

Files `runDCAnalysis.m`, `runTransientAnalysis.m`, and `runACAnalysis.m` allow one to run the 3 basic analysis algorithms on any circuit. This takes in a model (in this case a transistor model, like BSIM, MIT Virtual Source (MVS), or PSP). It builds a circuit with the compact model and another one with a STEAM approximation of the model. The scripts then run the corresponding analysis on the circuit and compare the results. Circuit netlists that are used for these can be found under `circuits` directory. The netlists are generic and can be used interchangeably with any of the analyses/transistor models. For example, by default, `runDCAnalysis.m` uses `daeMOSInverter.m`, which is a CMOS inverter circuit. The script also reports numerical error and speedup for using both splines and BLI as the interpolant.

9.2.5 BLI and Splines

`compare_BLI_w_Splines.m` in `ThesisExtras` reproduces the comparison on BLI and Splines over smooth functions described in Section 5.1. For this, a smooth function is chosen (`getTestFHandle()` creates a function handle with random parameters). First, BLI and spline interpolants are constructed with a few sample points, and then, they are reconstructed with a significantly higher number of sample points. In both the cases, then, the interpolants are compared with the original function for accuracy. One can change the interpolant arguments for BLI to be the same as Splines which uses uniform sample points. This brings out the instability in Chebyshev polynomials when using uniform sample points.

In the same script, you have the option to change the function class being used to perform the test. By default, this class is set to 'smooth'. Changing it to 'd' reproduces the results on a discontinuous function, and 'c' produces results for a continuous function (with discontinuous derivative).

9.2.6 ALACARTE: Circuit analyses

Results for device evaluation and elementary analyses can be obtained with the same scripts that were being used for STEAM. `runHB.m` in `examples` can be used to run Harmonic balance for both oscillators (a 3-stage ring oscillator is provided), and circuits driven by a periodic signal (a Gilbert cell circuit is provided).

9.2.7 Sparse Measurements

File `BSIM_2D_SVD.m` can be used to get a 2D slice of BSIM data, and perform Singular Value Decomposition (SVD) on it. First, the singular values are produced for the data slice, and then, the reconstruction error is evaluated for using the k highest singular values. `experiment_BSIM_DCT.m` can

reproduce the results for reconstruction with a partial set of DCT coefficients for 1-dimensional device data. It also demonstrates reconstruction of the device characteristics using sparse measurements in 1-dimension. The comparison between SVD and DCT can be generated using `SVD_vs_DCT.m`.

9.2.8 Application of CS

`phantom_experiment.m` can be used to generate the fourier coefficients for the Phantom image along radial lines in the 2-dimensional Fourier domain. Since the results for reconstructing the image are readily available [24], they have not been added. Readers can, however, experiment with the reconstruction of an image for which some of the pixels are available. This can be done with the script `image_reconstruction.m`.

Application of this method for a 1-dimensional device model is also included in `experiment_BSIM_DCT.m` previously mentioned.

9.2.9 Model Diagnosis

To highlight the issues related to model discontinuities in Radio Frequency (RF) design, runHB-Sweep.m can be used. This script takes a circuit driven by a periodic signal (by default it picks the source follower circuit), and performs a DC sweep on the input voltage. At each steady-state operating point, small signal analysis is performed using Harmonic Balance (HB) and the transfer function's 1^{st} and 2^{nd} harmonic are reported. The script titled `diode_connected_mosfet.m` reproduces the results shown for a 1-dimensional MOSFET model in Section 7.1.