NORTH-HOLLAND

Multiplication-Free Evaluation of Polynomials via a Stochastic Bernstein Representation

Chi-Chin Chou and William A. Sethares

Department of Electrical and Computer Engineering
University of Wisconsin-Madison
1415 Johnson Dr.
Madison, Wisconsin 53706
chou@cae.wisc.edu, sethares@ece.wisc.edu

ABSTRACT

A new multiplication-free method for the evaluation of multidimensional polynomials is proposed. The method is based on a Stochastic Bernstein Representation (SBR) and utilizes a random number generator, a locally encoded data structure, and a system table. The SBR is shown to be capable of representing any polynomial function and of approximating any continuous function arbitrarily closely. An error bound analysis is performed using a large deviations technique. A variant of the SBR, which has been used by others to balance an inverted pendulum, is also analyzed.

1. INTRODUCTION

Polynomial evaluation is needed in many scientific and engineering disciplines. In computer aided geometric design (CAGD), powerful multidimensional polynomial representations are crucial to many aspects of object modelling [1]. In control and communication, simple and efficient polynomial evaluation is important for real time implementation of nonlinear controllers. Although a one-dimensional polynomial can always be simply represented by a look-up table, this is not practical in higher dimensions because memory storage requirements increase exponentially with the dimension. On the other hand, evaluation of a polynomial by straightforward

APPLIED MATHEMATICS AND COMPUTATION 79:1–25 (1996)

655 Avenue of the Americas, New York, NY 10010
0096-3003/96/$15.00
SSDI 0096-3003(95)00156-C

multiplication requires considerable computation time and is often impossible in real time applications. For some special polynomial forms, iterative methods of computation are available (such as the Horner method for the power form [2, 3, 4, 5] or the de Casteljau algorithm for the Bernstein form [6]). But in all of these methods, either multiplication is indispensable, or the algorithm cannot be parallelized in more than one dimension.

This paper proposes a new method of multiplication-free evaluation of polynomials which can be readily implemented in parallel. The method, which we call the Stochastic Bernstein Representation (SBR), is based on a random coding of the input and a mapping that is fundamentally based on a neighborhood, or local structure. We show that the SBR can converge to any continuous function almost surely in the pointwise sense, and an error bound is given that relates the required length of the input/output array tables and the final error in the functional representation. A variant of the SBR is also analyzed which explains and justifies an algorithm that has been used to balance an inverted pendulum by Lee *et al.* [7, 8]. We show that the success of this variant of the SBR (called a "Stochastic Cellular Automata" in these papers) is based on a polynomial approximation to the solution of the underlying nonlinear dynamical system.

Any one-dimensional n^{th} order polynomial can be represented in a standard power form as $\sum_{i=0}^{n} a_i x^i$, or it can be represented in "Bernstein form" as $\sum_{i=0}^{n} b_i x^i (1-x)^{n-i}$ for all $x \in (0,1)$. The sets of coefficients a_i and b_i are in one to one correspondence and can be transformed readily as [19]

$$b_k = \sum_{j=0}^{k} \binom{k}{j} \binom{n}{j}^{-1} a_j,$$

$$a_k = \sum_{j=0}^{k} (-1)^{k-j} \binom{n}{k} \binom{k}{j}^{-1} b_j.$$

This Bernstein polynomial form provides the basis on which the SBR is built.

In Section 2, the one-dimensional SBR is proposed. Section 3 shows that the SBR can approximate any continuous function almost surely in the pointwise sense. In Section 4, a large deviation technique is used to analyze the convergence rate and to estimate the required length of the input/output array for a given approximation error. Section 5 generalizes the one-dimensional SBR to multiple dimensions. In Section 6, several issues are

briefly discussed: First, the random number generator can be profitably implemented by an appropriate Cellular Automata (CA) which is further detailed in [20], second, the "Stochastic Cellular Automata" of Lee *et al.* is analyzed, and third, the parameter estimation problem for the system table is discussed. Conclusions appear in the final section.

2. THE STOCHASTIC BERNSTEIN REPRESENTATION IN ONE DIMENSION

Consider the following representation of a function as shown in Figure 1. Let the input x take values between 0 and 1 and let y be the output. Select an odd integer n that defines the local neighborhood size, and an integer l that defines the resolution or fineness of the representation. The SBR representation is composed of three parts.

Input array: The input array is a binary l vector. Each element of the input array is assigned 1 with probability x, and 0 with probability $1 - x$.

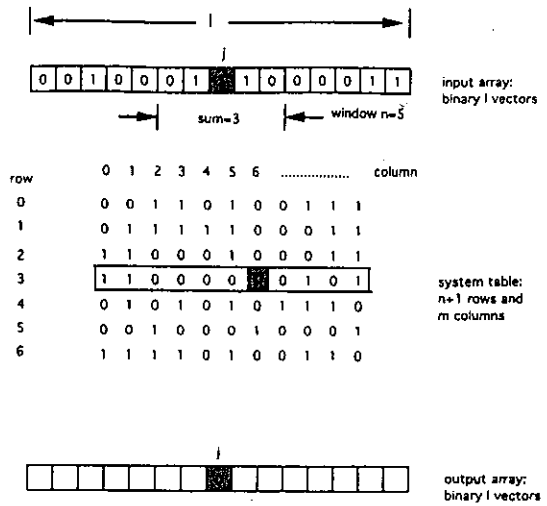

FIG. 1. The Stochastic Bernstein Representation as illustrated by this mapping of the j^{th} bit of the input array. The sum of elements in the $n = 5$ window around the j^{th} bit (sum = 3 in this case) points to the third row of the system table. A random number (in this case 6) is chosen, and the value of the element in row 3, column 6 is mapped to the j^{th} bit of the output array.

System table: The system table is an $n+1$ by m binary matrix. The ratio of the number of 0's and 1's in each row is used to determine the coefficients of the polynomial to be approximated, and the value of m determines the resolution (fineness of representation) of the coefficients.

Output array: The output array is a binary l vector whose elements are determined from the system table by the mapping mechanism described below. The output value $y(x)$ is defined to be the normalized sum of the elements of the output array.

Mapping mechanism: The value of the j^{th} element of the output array is determined as follows. The sum of the n nearest neighbors of the j^{th} element of the input array (call this sum s) is used as a pointer to row s of the system table. Note that the value of the sum s ranges from 0 to n, and thus there are $n+1$ rows in the system table. The value of the j^{th} element of the output array is then a copy of some element (which is chosen randomly) from the s^{th} row of the system table. The complete SBR functional form can be written clearly in pseudo code:

Define:

$IA[k]$ = k^{th} element of the input array.
$ST[i, j]$ = value of the element of the system table in the i^{th} row and j^{th} column.
$OA[k]$ = k^{th} element of the output array.
$Random(0, h)$ = a uniform random number between 0 and h.
$Integer(r)$ = integer part of a real number r.

Initialize the input array:

for $k = 1, \ldots, l$
$r1 = Random(0, 1)$
$IA[k] = \begin{cases} 1 & \text{if } r1 > x \\ 0 & \text{if } r1 \leqslant x \end{cases}$
end k

Mapping mechanism:

for $j = 1, \ldots, l$
$s = \sum_{k=(j-n/2)_{mod\, l}}^{(j+n/2)_{mod\, l}} IA[k]$
$r2 = Integer(Random(0, m))$
$OA[j] = ST[s, r2]$
end j

Calculate the output value:

$$y(x) = (1/l)\sum_{j=1}^{l} OA[j].$$

REMARKS.

(1) The *mod* function in the calculation of s essentially turns the intput array into a "circular buffer" and allows all elements j to have n neighbors.

(2) The accuracy for which a given input x and its corresponding output $y(x)$ can be represented is dependent on the number of elements l in the input array. Larger l implies more accurate representation.

(3) The complexity of the mapping between x and $y(x)$ is dependent on the number of neighbors n. Larger n implies more complex mappings.

Concrete results of the accuracy of representations and the complexity of representations as functions of n and l are given in the following sections.

The following examples illustrate how the SBR can be used to represent (or approximate) functions. The system tables are found by expanding the functions in a basis formed by Legendre polynomials and then transforming to find the coefficients of the Bernstein polynomials. In the figures, the solid line is the original function, the squiggly line is the SBR approximation.

EXAMPLE 1. Choose $n = 5$ and $m = 100$ to approximate $f(x) = 0.151\sin(2\pi x) + 0.5$. With $l = 5000$, plots of $f(x)$ and the SBR approximation are shown in Figure 2a. The error is shown in Figure 2b. With

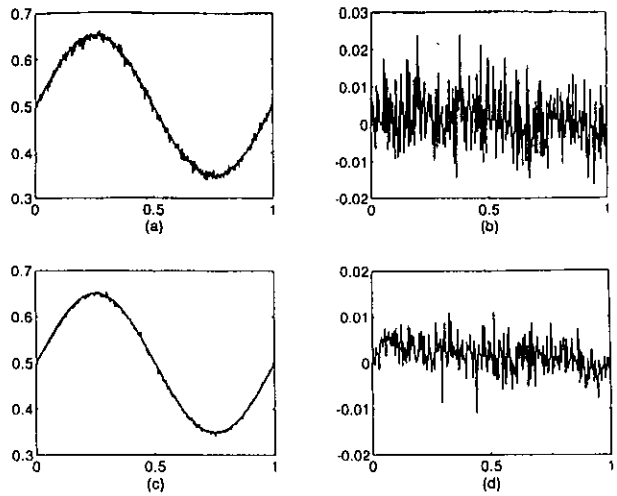

FIG. 2. Approximations to $f(x) = 0.151\sin(2\pi x) + 0.5$. (a) $f(x)$ and the $l = 5000$ approximation. (b) error between $f(x)$ and the $l = 5000$ approximation. (c) $f(x)$ and the $l = 20000$ approximation. (d) error between $f(x)$ and the $l = 20000$ approximation.

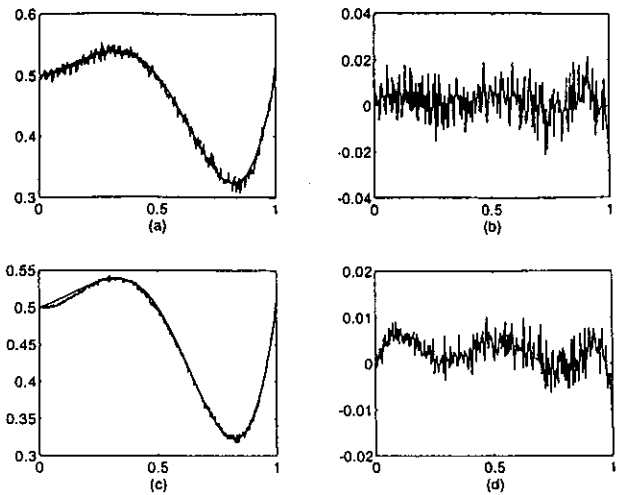

FIG. 3. Approximations to $f(x) = 0.0167\sin(2\pi x)\exp(3x) + 0.5$. (a) $f(x)$ and the $l = 5000$ approximation. (b) error between $f(x)$ and the $l = 5000$ approximation. (c) $f(x)$ and the $l = 20000$ approximation. (d) error between $f(x)$ and the $l = 20000$ approximation.

$l = 20000$, Figure 2c shows the approximation and Figure 2d shows the error. As l increases, the error decreases.

EXAMPLE 2. Choose $n = 5$ and $m = 100$ to approximate $f(x) = 0.0167\sin(2\pi x)\exp(3x)$. With $l = 5000$, plots of $f(x)$ and the SBR approximation are shown in Figure 3a. The error is shown in Figure 3b. With $l = 20000$, Figure 3c shows the approximation and Figure 3d shows the error. Again, note how the error decreases as l increases.

3. CONVERGENCE

This section shows that the SBR can be used to approximate any continuous function almost surely in the pointwise sense.

Define an equivalence class $Q_n(x)$ to consist of all order n polynomials which can be derived from each other by scaling and biasing, i.e., $p(x)$ and $q(x)$ are both members of a single class $Q_n(x)$ if and only if there exist c and d, such that $p(x) = cq(x) + d$.

The first lemma shows that the expected value of any SBR of order n (where n is the number of neighbors used to calculate the pointer into the system table) is capable of representing any polynomial of order n, up to the equivalence class $Q_n(x)$. Since the SBR is stochastic in nature, the representation is only valid almost surely. Details are in the second lemma.

LEMMA 1. *Given any n^{th} order polynomial $p(x)$, let $Q_n(x)$ represent the class of order n polynomials that are equivalent to $p(x)$ under scaling and biasing. Then there is a SBR function $y(x)$ represented by a $(n+1) \times m$ system table, and a $q(x) \in Q_n(x)$ such that*

$$E[\,y(x)] = q(x) \quad \forall x \in [0,1].$$

PROOF. See the Appendix A.

As pointed out in Appendix A, the expected value of the j^{th} bit of the output array is

$$\bar{y}_j(x) = b_0(1-x)^n + b_1(1-x)^{n-1}x + \cdots + b_i(1-x)^{n-i}x^i + \cdots + b_n x^n, \tag{1}$$

where $b_i \equiv a_i C_i^n$ and a_i and C_i^n are defined in Appendix A, which is known as the Bernstein polynomial of order n [9].

Next, define the random variable

$$\overline{Y}(x) = \bar{y}_1(x) \quad \text{with probability 1.}$$

LEMMA 2. *Let $\hat{y}_l(x) \equiv (1/l)\sum_{i=1}^{l} y_i(x)$. Then with fixed n, $\hat{y}_l(x) \to \overline{Y}(x)$ a.s. for all $x \in [0,1]$ as $l \to \infty$.*

PROOF. Since n is finite and $x \in [0,1]$, $\bar{y}_1(x)$ is finite by (1). By the Strong Law of Large Numbers for the i.i.d. case [10], the lemma follows. ■

Thus, the two lemmas show that when $l \to \infty$, any polynomial up to order n (or its equivalent after scaling and biasing) can be approximated almost surely in the pointwise sense by the SBR with a system table containing $n+1$ rows. Finally, letting n increase provides the main result that any continuous function can be approximated by a suitable SBR.

THEOREM 1. *Given any continuous function $f(x)$ defined on $[0,1]$, let $W(x)$ represent the class of continuous functions that are equivalent to $f(x)$ under scaling and biasing. Then there is a sequence of SBR functions $y_n(x)$'s represented by $(n+1) \times m$ system tables, and a $g(x) \in W(x)$ such that*

$$y_n(x) \to g(x) \quad a.s. \; \forall x \in [0,1] \text{ as } n \to \infty.$$

PROOF. This theorem follows by the above two lemmas and the Weierstress-Stone Theorem [11]. ■

4. ERROR BOUND

Since the SBR is a stochastic approximation of a deterministic function, there is always some error. The two examples of Section 2 suggest that this error decreases as l increases. This section gives a bound on the error in terms of the probability of exceeding a certain value. The analysis supposes that a_i's are represented exactly (i.e., m is infinite) and derives the bound by applying a result from large deviations.

Consider the sample average of i.i.d. random variables

$$\bar{y}_l = \frac{y_1 + y_2 + \cdots + y_l}{l}$$

where $\{y_i\}$, $i = 1, 2, \ldots$ are i.i.d. with finite mean m_y. Consider the function

$$I(t) \equiv \sup_\theta [\theta t - \log M(\theta)],$$

where θ is a real number and

$$M(\theta) \equiv E[\exp(\theta y_1)].$$

$M(\theta)$ is called the *moment generating function*, and $I(t)$ is called the *large deviation rate function*. The following properties of $I(t)$ are known, and proofs can be found in [12].

PROPERTY 1. *$I(t)$ is convex.*

PROPERTY 2. *$I(t)$ has its minimum value at $t = m_y$. Furthermore, $I(m_y) = 0$.*

The main large deviation result is from Cramér:

THEOREM 2. *Assume that $M(\theta) < \infty$ for all θ. Then for every closed subset $F \subset R$,*

$$\limsup_{n \to \infty} \frac{1}{l} \log P\{\bar{y}_l \in F\} \leqslant - \inf_{t \in F} I(t). \tag{2}$$

For the SBR in Section 1, let $A(x)$ represent Equation (1) which is the expected value of any output bit $y_i(x)$. Notice that any output bit $y_i(x)$ is either 1 or 0. Then

$$\begin{aligned} M(\theta) &= E[\exp(\theta y_i(x))] \\ &= e^\theta A(x) + 1 - A(x). \end{aligned}$$

Consider $F = [A(x) + \varepsilon, 1]$, where ε is a small positive number. By the two properties of $I(t)$,

$$\begin{aligned} \inf_{t \in F} I(t) &= I(A(x) + \varepsilon) \\ &= \sup_\theta \{\theta(A(x) + \varepsilon) - \log[e^\theta A(x) + 1 - A(x)]\}. \end{aligned} \tag{3}$$

Since the term in the curly brace of (3) is a continuous function of θ, the supremum becomes the maximum which occurs at

$$\theta^*(x) = \log\left[\frac{(1 - A(x))(A(x) + \varepsilon)}{A(x)(1 - (A(x) + \varepsilon))}\right],$$

and the maximum value is

$$\begin{aligned} I(A(x) + \varepsilon) = &\log\left[\frac{(1 - A(x))(A(x) + \varepsilon)}{A(x)(1 - (A(x) + \varepsilon))}\right](A(x) + \varepsilon) \\ &- \log\left[\frac{1 - A(x)}{1 - (A(x) + \varepsilon)}\right]. \end{aligned} \tag{4}$$

Note that (4) is a function of x. An example shows how to apply this result.

EXAMPLE 3. What value of l is needed in the SBR to approximate the polynomial

$$p(x) = 0.5024135C_0^5(1-x)^5 + 0.665843C_1^5(1-x)^4 x + C_2^5(1-x)^3 x^2 + 0.334157C_4^5(1-x)x^4 + 0.497586C_5^5 x^5,$$

(which is the 5^{th} order polynomial used to approximate $f(x) = 0.151\sin(2\pi x) + 0.5$ in Example 1), so that the largest approximation error to $p(x)$ over $x \in [0,1]$ is less than 0.01 with probability 0.99 and with probability 0.999

Since the $I(\cdot)$ in (2) represents the rate of the exponential decay of $P\{\bar{y}_l \in F\}$ with $F = [A(x) + 0.01, 1]$ in this example, the global minimum of $I(\cdot)$ over $x \in [0,1]$ represents the "lowest" rate for $P\{\bar{y}_l \in F\}$ to decay to zero. Thus for a fixed n, the largest approximation error will occur at the x where $I(\cdot)$ achieves it global minimum. By setting the derivative of (4) to zero, the global minimum of $I(\cdot)$ is found to occur at $x = 0.998$ taking the value of $I(A(0.998) + 0.01) = 0.00020004$. (Actually, there are several x's in $[0,1]$ where the global minimum is achieved in this example). To achieve an approximation error of less than 0.01 with probability 0.99, this implies that $P\{\bar{y}_l \in F\} \leqslant (1 - 0.99) = 0.01$ at $x = 0.998$. By (2) and (4)

$$\begin{aligned} P\{\bar{y}_l(x) \in F\} &\leqslant \exp[-lI(A(x) + 0.01)] \\ &\leqslant \exp[-lI(A(0.998) + 0.01)] \\ &= \exp[-0.00020004\, l] \\ &\leqslant 0.01. \end{aligned}$$

Thus, l needs to be larger than 23025. Using an identical argument, an error of less than 0.01 with probability 0.999 requires l larger than 34538.

REMARK. Since the resolution of the representations of the a_i's (i.e., m) in any real SBR is finite, the required l may be a little larger. On the other hand, the above calculation of the error bound is conservative (since it bounds the error at the "worst" x, and the lim sup is used in (2)), the required l may actually be a little smaller. This can be seen in the Figure 2, where there is only one point out of 300 points in $[0,1]$ with error greater than 0.01, even though $l = 20000$. Thus, the error bound derived here can

only serve as an approximation to the required length of the input/output array.

5. MULTI-DIMENSIONAL SBR

The two-dimensional SBR with (n_x, n_y) neighborhood structure is shown in Figure 4. The encoding of the inputs and decoding of the output are analogus to the one-dimensional SBR. The mapping mechanism is also similar to the one-dimensional SBR. The j^{th} element of the input array is mapped to the $(s_2(n_x + 1) + s_1)^{th}$ row of the system table where the sum of the neighbors around the j^{th} element of the x input array is equal to s_1, and the sum of neighbors around the j^{th} element of the y input array is equal to

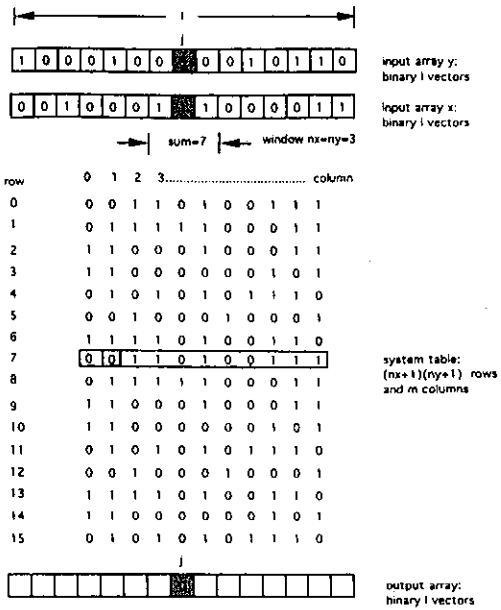

FIG. 4. The 2-D Stochastic Bernstein Representation is illustrated by this mapping of the j^{th} bit of the input array. The sum of elements in the $n_x = 3$ window around the j^{th} bit of the input array x ($s_1 = 3$ in this case) and the sum of elements in the $n_y = 3$ windows around the j^{th} bit of the input array y ($s_2 = 1$ in this case) points to the row seven of the system table by the formula: $sum = s_2(n_x + 1) + s_1$. sum = 7 in this case. A random number (in this case 1) is chosen, and the value of the element in row 7, column 1 is mapped to the j^{th} bit of the output array.

s_2. Thus, there are $(n_x + 1)(n_y + 1)$ rows in the system table. The value of the j^{th} element of the output array is a copy of some element of the $(s_2(n_x + 1) + s_1)^{th}$ row of the system table, which is chosen randomly. This may be represented in pseudo-code as:

Define:

$IAX[k] = k^{th}$ element of the input array x.
$IAY[k] = k^{th}$ element of the input array y.
$ST[i, j]$ = value of the element of the system table in the i^{th} row and j^{th} column.
$OA[k] = k^{th}$ element of the output array.
$Random(0, h)$ = a uniform random number between 0 and h.
$Integer(r)$ = integer part of a real number r.

Initialize the input arrays:

for $k = 1, \ldots, l$
$r1 = Random(0, 1)$
$r2 = Random(0, 1)$

$$IAX[k] = \begin{cases} 1 & \text{if } r1 > x \\ 0 & \text{if } r1 \leqslant x \end{cases}$$

$$IAY[k] = \begin{cases} 1 & \text{if } r2 > y \\ 1 & \text{if } r2 \leqslant y \end{cases}$$

end k.

Mapping mechanism:

for $j = 1, \ldots, l$
$s = (n_x + 1)\sum_{k=(j-n_y/2)_{mod\,l}}^{(j+n_y/2)_{mod\,l}} IAY[k] + \sum_{k=(j-n_x/2)_{mod\,l}}^{(j+n_x/2)_{mod\,l}} IAX[k]$
$r3 = Integer(Random(0, m))$
$OA[j] = ST[s, r3]$
end j.

Calculate the output value:

$y(x) = (1/l)\sum_{j=1}^{l} OA(j)$.

It is easy to show (as in Lemma 4) that the expected value of the output is

$$\bar{z}(x, y) = \sum_{i=0}^{n_x} \sum_{j=0}^{n_y} a_{ij} C_i^m (1 - x)^{n_x - i} x^i C_j^n (1 - y)^{n_y - j} y^j. \qquad (5)$$

Comparing this polynomial with the power form

$$\sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} x^i y^j$$

shows that there is a one-to-one correspondence between the sets of a_{ij}'s and b_{ij}'s with an appropriate scaling and biasing. Similar lemmas and theory as in Section 2 can be derived for the two-dimensional SBR, and generalizations to higher dimensional SBR's are straightforward.

EXAMPLE 4. Choose $n_x = n_y = 3$ and $m = 100$ to approximate the two dimensional polynomial (5) where the parameters

$$a_{00} = 0.1 \qquad a_{02} = 0.1$$
$$a_{10} = 0.2 \qquad a_{12} = 0.8$$
$$a_{20} = 0.3 \qquad a_{22} = 0.7$$
$$a_{30} = 1.0 \qquad a_{32} = 1.0$$
$$a_{01} = 0.4 \qquad a_{03} = 0.3$$
$$a_{11} = 0.5 \qquad a_{13} = 0.2$$
$$a_{21} = 0.4 \qquad a_{23} = 0.1$$
$$a_{31} = 1.0 \qquad a_{33} = 1.0$$

were arbitrarily assigned. The results are plotted in Figure 5 and Figure 6. Figure 5a shows the 2-D function. Figure 5b shows the 2-D SBR approximation using $l = 5000$ and Figure 5c shows the approximation with $l = 20000$.

Figure 6a and Figure 6b show the parametric curve $c(t) = (0.5 \sin(2\pi t), 0.5\cos(2\pi t))$, for $t = [0, 1]$ on the 2-D surface and its two SBR approximations using $l = 5000$ and $l = 20000$ respectively. Figure 6c and Figure 6d show the curve $c(t) = (0.5\cos(t), \sin(t))$, for $t = [0, 1]$ on the 2-D surface and its two SBR approximations using $l = 5000$ and $l = 20000$ respectively. Note that as l increases, the error decreases in all approximations.

For a fixed l (the length of the input arrays) and with a fixed random number generator, the size of the memory required to store the input array for the k dimensional SBR is k times of the size of the input array for the one dimensional SBR. For a small number of dimensions, the storage required in the system table is inconsequential, though for larger dimensions this grows as $(n_{max} + 1)^k$, where k is the dimension) and n_{max} is the

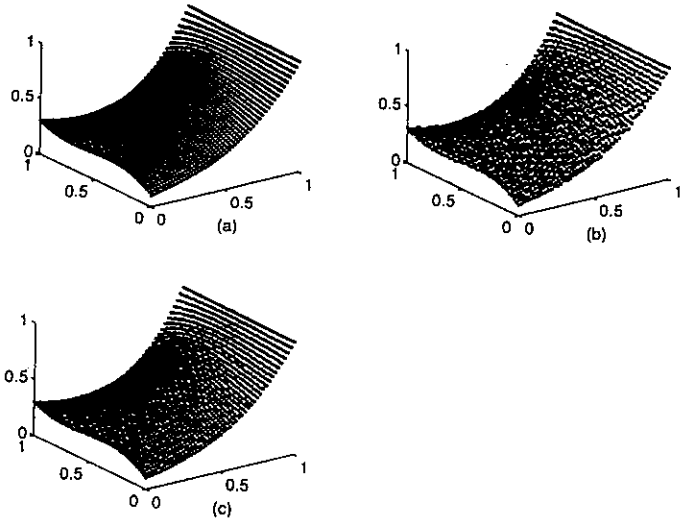

FIG. 5. Approximations to the 2-D function $z(x, y)$ of Example 3. (a) the function $z(x, y)$. (b) the $l = 5000$ approximation. (c) the $l = 20000$ approximation.

maximum order of the polynomial to be represented. While this is exponential growth, it contrasts strikingly with storage requirements for a system look-up table, in which growth in memory is proportional to r^k, where r is the number of points needed to represent the function. Since the number of points r needed to represent a given polynomial is much greater than the order n of the polynomial, $r \gg n$, the storage is greatly reduced by a SBR.

6. OTHER ISSUES

6.1. *Random Number Generator*

In the SBR, the most complicated computations occur in the random number generator (RNG). Thus a simple, reliable, and parallelizable RNG is crucial for practical implementations. It has been reported in several papers [13, 14] that certain cellular automata (CA) can be used as good parallel RNG's. A cellular automata evolves in discrete time with the next value of one site determined by its previous value and those of its neighbors. It has been demonstrated that a rule 30 CA or a hybrid of rules 90 and 150 CA are very good RNG and can be easily implemented in parallel [14]. Thus, in a practical, parallel implementation of the SBR, the input array, system table,

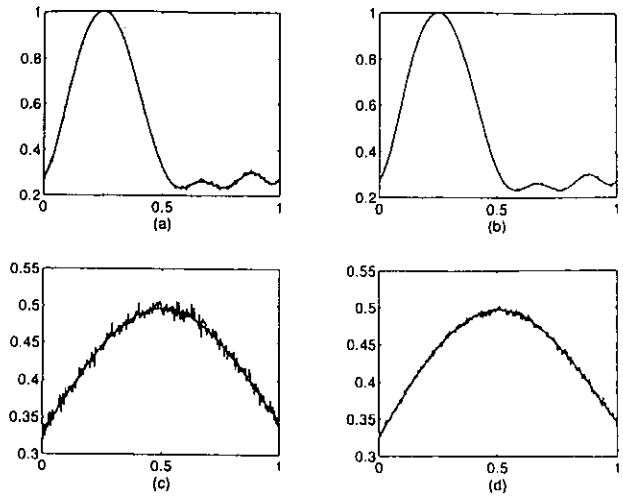

FIG. 6. Approximations to curves on surface of the 2-D function $z(x, y)$ of Fig. 5a. (a) $c(t) = (0.5\sin(2\pi t), 0.5\cos(2\pi t))$ for $t = [0, 1]$ and the $l = 5000$ approximation. (b) $c(t) = (0.5\sin(2\pi t), 0.5\cos(2\pi t))$ for $t = [0, 1]$ and the $l = 20000$ approximation. (c) $c(t) = (0.5\cos(t), sin(t))$ for $t = [0, 1]$ and the $l = 5000$ approximation. (d) $c(t) = (0.5\cos(t), sin(t))$ for $t = [0, 1]$ and the $l = 20000$ approximation.

and output array could be augmented by the addition of a suitable CA based RNG. Practical parallel implementation is discussed in [20].

6.2. *A Variant of SBR and Its Application to the Pendulum Balancing Problem*

In the paper [8] Lee *et al.* use a slightly different encoding mechanism (which they called a Stochastic Cellular Automata (SCA)) to balance an inverted pendulum. Instead of assigning values to the bits of the input array based on a probability distribution determined by the value of the input x, they force the number of 1's in the input array to be precisely equal to $[lx]$, where $[z]$ represents the integer part of z. Although they succeeded in balancing the pendulum, they were unable to explain how and why the scheme was successful. We show in Appendix B that with the modification of the encoding of the input array, the (modified) SBR can still be used to approximate any continuous function. Thus, the success of Lee's scheme may be interpreted as being based on a polynomial approximation to the solution of the nonlinear dynamic system equations underlying the inverted pendulum. Notice that such an encoding scheme of the input array cannot be implemented in parallel.

6.3. Specification of the System Table

The number of rows in the system table is determined by the neighborhood structure, and the width of each row is directly related to the resolution of the coefficients a_i. There are many ways to find the best a_i's to approximate a given known continuous function. In Examples 1 and 2, the coefficients were found by expanding the given function in the Legendre polynomial form and by then comparing the coefficients of the Bernstein polynomial with those of the Lengendre polynomial. One can also view the

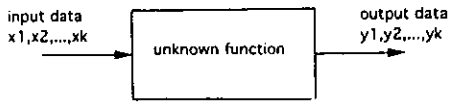

FIG. 7. Approximation of an unknown function via the Bernstein polynomial basis, $B_0(x), B_1(x), \ldots, B_n(x)$, where

$$B_i(x) \equiv a_i C_i^n (1-x)^{n-i} x^i,$$

of order n can be accomplished by solving the normal equation for the parameters $a_0, a_1, \ldots, a_n$ in the following matrix form.

$$\begin{bmatrix} \sum_{j=1}^{k} B_0(x_j) B_0(x_j) & \sum_{j=1}^{k} B_0(x_j) B_1(x_j) & \cdots & \sum_{j=1}^{k} B_0(x_j) B_n(x_j) \\ \sum_{j=1}^{k} B_1(x_j) B_0(x_j) & \sum_{j=1}^{k} B_1(x_j) B_1(x_j) & \cdots & \sum_{j=1}^{k} B_1(x_j) B_n(x_j) \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{j=1}^{k} B_n(x_j) B_0(x_j) & \sum_{j=1}^{k} B_n(x_j) B_1(x_j) & \cdots & \sum_{j=1}^{k} B_n(x_j) B_n(x_j) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{j=1}^{k} B_0(x_j) y_j \\ \sum_{j=1}^{k} B_1(x_j) y_j \\ \vdots \\ \sum_{j=1}^{k} B_n(x_j) y_j \end{bmatrix},$$

where $x_1, x_2, \ldots, x_k$ are input data and $y_1, y_2, \ldots, y_k$ are output data.

specification of the system table as a form of "system identification" in which a given set of input/output data is to be encoded by the SBR. The problem then reduces to finding a set of a_i which best match the input/output data. There are many criteria which can be used to determine an optimal set of a_i's. For instance, the L^2 criterion is common in many data fitting tasks which apply some form of the "projection theorem" by which a normal equation of the coefficients a_i can be derived [15] as shown in Figure 7. For other kinds of approximations such as L^1 or L^∞, a nonlinear equation has to be solved and there may be local minimum [16, 17]. There are also general heuristic methods that can be used to find the coefficients such as simulated annealing and the genetic algorithm [18]. Simple methods such as bit-flipping [8] may also be useful.

7. CONCLUSIONS

This paper proposed a new method (the so-called Stochastic Bernstein Representation or SBR) for multiplication-free evaluation of multidimensional polynomials. The SBR uses a random number generator, a local rule to encode the input, and a system table for calculation of the output. The scheme is suitable for parallel implementation. Compared to simple table look-up of a polynomial functions, the SBR requires for less the memory in higher dimensional cases. The SBR is much faster than direct evaluation because it requires no multiplications. Future extensions of this work include VLSI implementation of the SBR and detailed application to real problems.

APPENDIX A: PROOF OF LEMMA 4.

Given $x \in [0, 1]$, consider the j^{th} bit of the input array. The probability that the j^{th} bit is mapped to the i^{th} row of the system table is

$$\begin{aligned} \text{probability of row } 0 &= C_0^n (1-x)^n \\ \text{probability of row } 1 &= C_1^n (1-x)^{n-1} x \\ &\vdots \\ \text{probability of row } i &= C_i^n (1-x)^{n-i} x^i \\ &\vdots \\ \text{probability of row } n &= C_n^n x^n, \end{aligned}$$

where $C_i^n \equiv n!/i!(n-i)!$. Let a_i be the percentage of 1's in the i^{th} row of the system table and thus $a_i \in [0, 1]$. Then the expected value of the j^{th} bit of the output array is

$$\bar{y}_j(x) = a_0 C_0^n (1-x)^n + a_1 C_1^n (1-x)^{n-1} x + \cdots + a_i C_i^n (1-x)^{n-i} x^i + \cdots + a_n C_n^n x^n, \tag{6}$$

which is again the Bernstein polynomial form as in (1).

To make a comparison between an n^{th} order polynomial in power form with the Bernstein form (6), let

$$p_n(x) \equiv a_0 C_0^n (1-x)^n + a_1 C_1^n (1-x)^{n-1} x + \cdots + a_i C_i^n (1-x)^{n-i} x^i + \cdots + a_n C_n^n x^n$$

and

$$p'_n(x) \equiv d_0 C_0^n (1-x)^n + d_1 C_1^n (1-x)^{n-1} x + \cdots + d_i C_i^n (1-x)^{n-i} x^i + \cdots + d_n C_n^n x^n,$$

where $d_i = 2a_i - 1$ for all $i = 0, 1, \ldots, n$. Clearly, $d_i \in [-1, 1]$ and

$$p'_n(x) = 2p_n(x) - 1.$$

Thus $p_n(x)$ and $p'_n(x)$ are members of the same equivalence class $Q_n(x)$.

Suppose that $p'_n(x)$ can also be represented in power form as

$$p'_n(x) = m(b_0 + b_1 x + \cdots + b_n x^n) \quad \text{for all } x \in [0, 1], \tag{7}$$

where b_0 and m will be chosen later, and $b_i \in R$ for all $i = 1, 2, \ldots, n$ are given. Note b_0 and m only affect biasing and scaling, so all such $p'_n(x)$ are in the same $Q_n(x)$.

Now we show that the exists a unique solution of $(m, d_0, d_1, \ldots, d_n)$ for the above equation. Comparing the coefficients of both sides of (7) gives

$$\begin{aligned} mb_0 &= d_0 \\ mb_1 &= -d_0 C_1^n + d_1 C_1^n \\ mb_2 &= d_0 C_2^n - d_1 C_1^n C_1^{n-1} + d_2 C_2^n \\ &\vdots \quad \vdots \end{aligned} \tag{8}$$

From (8), $b_0 = d_0/m$. Furthermore, (8) can be written in matrix form as

$$\begin{bmatrix} 1 & 0 & 0 & \cdots \\ -C_1^n & C_1^n & 0 & \cdots \\ C_2^n & -C_1^n C_1^{n-1} & C_2^n & 0 \cdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} d_0/m \\ d_1/m \\ \vdots \\ d_n/m \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}. \tag{9}$$

The matrix (9) is invertible since it is lower triangular and none of the diagonal elements are zero. Let $(s_0, s_1, \ldots, s_n)$ be the solution for $(d_0/m, d_1/m, \ldots, d_n/m)$ and let $|s_{max}|$ be the maximum absolute value of the set $\{s_i\}$. Choose m to be $1/|s_{max}|$, then d_i is equal to $s_i m$ for all $i = 0, 1, \ldots, n$ which are in the range $[-1, 1]$ as required.

Since the mapping of the system table to each output bit is independent, the expected value of the output is

$$\bar{y}_l(x) = \frac{1}{l} \sum_{j=1}^{l} \bar{y}_j(x) = \bar{y}_1(x),$$

where l is the length of the input/output array. ■

APPENDIX B

This appendix shows that the variant of SBR (which we call SBR′) can also be used to approximate a continuous function by giving a similar result to Lemma 4 and 5, and Theorem 3 can easily be applied to this case, so these proofs are omitted.

Without loss of generality, let x be the $[lx]$ as defined in Section 6.

LEMMA 3. *Given any n^{th} order polynomial $p(x)$, let $Q_n(x)$ represent the class of order n polynomials that are equivalent to $p(x)$ under scaling and biasing. Then there is a SBR′ function $y(x)$ represented by a $(n+1) \times m$ system table, and a $q(x) \in Q_n(x)$ such that*

$$E[\, y(x)] = q(x) \quad \forall x \in (n+1, l-n).$$

PROOF. Suppose x is an integer such that $l - n > x > n + 1$ and let a_i be the percentage of 1's in the i^{th} row of the system table. The probability of any pattern (0's and 1's combination) of the input array given a value x can be calculated as

$$Prob\{\, any\ pattern \mid x\} = \frac{x!}{l(l-1)(l-2)\cdots(l-x+1)}.$$

Consider the j^{th} bit of the output array. The expected value $\bar{y}_j(x)$ can be calculated as

$$\begin{aligned}
&\bar{y}_j(x) \\
&= \frac{x!}{l(l-1)(l-2)\cdots(l-x+1)} \\
&\quad \times \left\{ a_0 C_0^n C_x^{l-n} + a_1 C_1^n C_{x-1}^{l-n} + \cdots + a_n C_n^n C_{x-n}^{l-n} \right\} \\
&= \frac{x!}{l(l-1)(l-2)\cdots(l-x+1)} \\
&\quad \times \left\{ a_0 C_0^n \frac{(l-n)(l-n-1)\cdots(l-n-x+1)}{x!} \right. \\
&\qquad + a_1 C_1^n \frac{(l-n)(l-n-1)\cdots(l-n-x+2)\,x}{x!} \\
&\qquad\quad \vdots \\
&\qquad + a_i C_i^n \frac{(l-n)(l-n-1)\cdots(l-n-x+i+1)\,x(x-1)\cdots(x-i+1)}{x!} \\
&\qquad\quad \vdots \\
&\left. + a_n C_n^n \frac{(l-n)(l-n-1)\cdots(l-x+1)\,x(x-1)\cdots(x-n+1)}{x!} \right\}
\end{aligned} \tag{10}$$

The i^{th} term in the brace of (10) multiplied by the term outside the brace can be written as

$$\begin{aligned}
&\frac{(l-n)(l-n-1)\cdots(l-n-x+i+1)\,x(x-1)\cdots(x-i+1)}{l(l-1)\cdots(l-x+1)} \\
&= \frac{(l-x)(l-x-1)\cdots(l-x-n+i+1)\,x(x-1)\cdots(x-i+1)}{l(l-1)\cdots(l-n+1)},
\end{aligned}$$

for $i = 0, \ldots, n-1$,

and the last (n^{th}) term is

$$\begin{aligned}
&\frac{(l-n)(l-n-1)\cdots(l-x+1)\,x(x-1)\cdots(x-n+1)}{l(l-1)\cdots(l-x+1)} \\
&= \frac{x(x-1)\cdots(x-n+1)}{l(l-1)\cdots(l-n+1)}.
\end{aligned}$$

Thus (10) can be simplified to

$$\begin{aligned}
\bar{y}_j(x) &= \frac{1}{l(l-1)\cdots(l-n+1)} \\
&\quad \times \left\{ a_0 C_0^n (l-x)(l-x-1)\cdots(l-x-n+1) \right. \\
&\qquad + a_1 C_1^n (l-x)(l-x-1)\cdots(l-x-n+2)\,x \\
&\qquad\quad \vdots \\
&\qquad + a_i C_i^n (l-x)(l-x-1)\cdots(l-x-+i+1)\,x(x-1)\cdots(x-i+1) \\
&\qquad\quad \vdots \\
&\qquad + a_{n-1} C_{n-1}^n (l-x)\,x(x-1)\cdots(x-n+2) \\
&\qquad \left. + a_n C_n^n x(x-1)\cdots(x-n+1) \right\}.
\end{aligned} \tag{11}$$

Define the polynomials,

$$
\begin{aligned}
S_0(x) &= (l-x)(l-x-1)\cdots(l-x-n+1)\\
S_1(x) &= (l-x)(l-x-1)\cdots(l-x-n+2)\,x\\
S_2(x) &= (l-x)(l-x-1)\cdots(l-x-n+3)\,x(x-1)\\
&\vdots\\
S_i(x) &= (l-x)(l-x-1)\cdots\\
&\qquad (l-x-n+i+1)\,x(x-1)\cdots(x-i+1)\\
&\vdots\\
S_{n-1}(x) &= (l-x)\,x(x-1)\cdots(x-n+2)\\
S_n(x) &= x(x-1)\cdots(x-n+1),
\end{aligned}
$$

where $S_i(x)$ is the polynomial term corresponding to $a_i C_i^n$ in the brace of (11). Then (11) can be rewritten

$$
\begin{aligned}
\bar{y}_j(x) = K(l,n)\{&d_0(n)S_0(x) + d_1(n)S_1(x) + \cdots\\
&+ d_i(n)S_i(x) + \cdots + d_n(n)S_n(x)\},
\end{aligned}
$$

where

$$
K(l,n) \equiv \frac{1}{l(l-1)\cdots(l-n+1)}
$$

and

$$
d_i(n) \equiv a_i C_i^n, \quad \text{for } i = 0,\ldots,n.
$$

Notice that each $S_i(x)$ is a polynomial of order n. Next, we want to show that these $n+1$ polynomials form a basis for any polynomial of order up to n.

Obviously, $S_0(x)$ and $S_1(x)$ are independent, i.e.,

$$
S_0(x) \neq \alpha S_1(x), \quad \forall x \in R \text{ for any constant } \alpha. \tag{12}
$$

Suppose that there exists a vector $(\alpha_0, \alpha_1, \ldots, \alpha_{i-1}) \in R^i$ such that for all x

$$
S_i(x) = \alpha_0 S_0(x) + \alpha_1 S_1(x) + \cdots + \alpha_{i-1} S_{i-1}(x). \tag{13}
$$

Below, we show that all α's must be zero which then contradicts the equality assumption (13).

Rewrite (13) as

$$
\alpha_{i-1}S_{i-1}(x) + \alpha_{i-2}S_{i-2}(x) + \cdots + \alpha_0 S_0(x) - S_i(x) = 0
$$

$$
\text{for all } x \in R \text{ and } i = 0,\ldots,n
$$

The above equation can be expanded as

$$
\begin{aligned}
&(l-x)(l-x-1)\cdots(l-x-n+i+1)\\
&\quad\times\{\alpha_{i-1}(l-x-n+i)\,x(x-1)\cdots(x-i+2)\\
&\quad+\alpha_{i-2}(l-x-n+i)(l-x-n+i-1)\,x(x-1)\cdots(x-i+3)\\
&\qquad\vdots\\
&\quad+\alpha_1(l-x-n+i)(l-x-n+i-1)\cdots(l-x-n+2)\,x\\
&\quad+\alpha_0(l-x-n+i)(l-x-n+i-1)\cdots(l-x-n+1)\\
&\qquad-x(x-1)\cdots(x-i+1)\} = 0.
\end{aligned}
$$

By the assumptions on (l, n, x), the product outside the brace is not zero. Thus

$$
\begin{aligned}
&\alpha_{i-1}(l-x-n+i)\,x(x-1)\cdots(x-i+2)\\
&\quad+\alpha_{i-2}(l-x-n+i)(l-x-n+i-1)\,x(x-1)\cdots(x-i+3)\\
&\qquad\vdots\\
&\quad+\alpha_1(l-x-n+i)(l-x-n+i-1)\cdots(l-x-n+2)\,x\\
&\quad+\alpha_0(l-x-n+i)(l-x-n+i-1)\cdots(l-x-n+1)\\
&\quad-x(x-1)\cdots(x-i+1) = 0.
\end{aligned} \tag{14}
$$

Since the constant term of (14) must be zero, $\alpha_0 = 0$. Thus (14) can be simplified to

$$\begin{aligned}
&\alpha_{i-1}(l - x - n + i)\,x(x-1)\cdots(x-i+2)\\
&\quad + \alpha_{i-2}(l - x - n + i)(l - x - n + i - 1)\,x(x-1)\cdots(x-i+3)\\
&\quad \vdots\\
&\quad + \alpha_1(l - x - n + i)(l - x - n + i - 1)\cdots(l - x - n + 2)\,x\\
&\quad - x(x-1)\cdots(x-i+1) = 0. \qquad (15)
\end{aligned}$$

Now the x can be pulled out from each term of (15) and the remainder must be zero. By defining $x' = x - 1$, the remainder of (15) can be written as

$$\begin{aligned}
&\alpha_{i-1}(l - x' - n + i - 1)\,x'(x'-1)\cdots(x'-i+3)\\
&\quad + \alpha_{i-2}(l - x' - n + i - 1)(l - x' - n + i - 2)\\
&\quad \times x'(x'-1)\cdots(x'-i+4)\\
&\quad \vdots\\
&\quad + \alpha_1(l - x' - n + i - 1)(l - x' - n + i - 2)\cdots(l - x' - n + 1)\\
&\quad - x'(x'-1)\cdots(x'-i+2) = 0. \qquad (16)
\end{aligned}$$

By the same reasoning, α_1 must be zero. Thus, $\alpha_0, \alpha_1, \ldots, \alpha_{i-1}$ are all zero, and $S_i(x)$ is independent of $S_0(x), S_1(x), \ldots, S_{i-1}(x)$ for all $i = 0, 1, \ldots, n$. This means that $S_i(x)$'s for $i = 0, 1, \ldots, n$ form a basis for polynomials of order n. By the independence of each output bit, the expected value of the output $\bar{y}(x) \equiv (1/l)\sum_{j=1}^{l} y_j(x)$ is equal to $\bar{y}_j(x)$.

With fixed l, n and by choosing appropriate $a_0, a_1, \ldots, a_n$, we can approximate any polynomial order n (up to a biasing and scaling) by a singular argument to that given in Appendix A. ■

REFERENCES

1 W. Bohm, A survey of curve and surface methods in CAGD, *Computer Aided Geometric Design 1*, pp. 1–60 (1984).

2 M. L. Dowling, A fast parallel Horner algorithm, *SIAM J. on Computing*, **19** (1):133–142 (1992).

3 W. Burleson, Polynomial evaluation in VLSI using distributed arithmetic, *IEEE Tran. on Circuits and Systems*, **37**(10):(1990).

4 H. Schroder, Top-down designs of instruction systolic arrays for polynomial interpolation and evaluation, *J. of Parallel and Distributed Computing* **6**:692–703 (1989).

5 E. R. Hansen, M. L. Patrick, and L. C. Richard, Polynomial evaluation with scaling, *ACM Trans. on Mathematical Software*, **16**(1):86–93, (1990).

6 L. L. Schumaker and W. Volk, Efficient evaluation of multivariate polynomials, *Computer Aided Geometric Design 3*, pp. 149–154 (1986).

7 Y. C. Lee, S. Qian, R. D. Jones, C. W. Barnes, G. W. Flake, M. K. O'Rourke, K. Lee, H. H. Chen, G. Z. Sun, Y. Q. Zhang, D. Chen, C. L. Giles, Adaptive stochastic cellular automata: theory, *Physica D* (45):159–180 (1990).

8 Y. C. Lee, S. Qian, R. D. Jones, C. W. Barnes, G. W. Flake, M. K. O'Rourke, K. Lee, H. H. Chen, G. Z. Sun, Y. Q. Zhang, D. Chen, C. L. Giles, Adaptive stochastic cellular automata: application, *Physica D* (45):181–188 (1990).

9 G. G. Lorentz, *Bernstein Polynomials*, Chelsea Publishing Company, 1986.

10 R. B. Ash, *Real Analysis and Probability*, Academic Press, 1972.

11 P. J. David, *Interpolation and Approximation*, Blaisbell Publishing Company, 1963.

12 J. A. Bucklew, *Large Deviation Techniques in Decision, Simulation, and Estimation*, Wiley Interscience Publishing, 1990.

13 S. Wolfram, Universality and complexity in cellular automata, *Physica D*, pp. 1–35 (1984).

14 P. D. Hortensins, R. D. Mcleods, H. C. Card, Parallel random number generator for VLSI systems using cellular automata, *IEEE Trans on Computers* **38**(10):1466–1473 (1989).

15 D. G. Guest, *Numerical Method of Curve Fitting*, Cambridge, 1961.

16 J. R. Rice, *The Approximation of Functions*, Addison-Wesley Publishing Company, 1964.

17 E. W. Cheney, *Introduction to Approximation Theory*, McGraw-Hill, 1966.

18 D. E. Goldberg, *Genetic Algorithm in Search, Optimization, and Machine Learning*. Addison-Wesley, New York, 1989.

19 R. T. Farouki and V. T. Rajan, On the numerical condition of polynomials in Berstein form, *Computer Aided Geometric Design* **4**:191–216 (1987).

20 Chi-Chin Chou, *Optimization via local interactions: Applications to the Steiner problem and rapid polynomial investigation*, Ph.D. Thesis, University of Wisconsin, August 1995.