A Matrix Method for Efficient Computation of Bernstein Coefficients*

Shashwati Ray
Systems and Control Engineering Group,
Room 114A, ACRE Building,
Indian Institute of Technology, Bombay, India 400 076
shashwatiray@yahoo.com

P. S. V. Nataraj
Systems and Control Engineering Group,
Room 114A, ACRE Building,
Indian Institute of Technology, Bombay, India 400 076
nataraj@sc.iitb.ac.in

Abstract

We propose a generalized method, called as the *Matrix* method, for computation of the coefficients of multivariate Bernstein polynomials. The proposed Matrix method involves only matrix operations such as multiplication, inverse, transpose and reshape. For a *general* box-like domain, the computational complexity of the proposed method is $O(n^{l+1})$ in contrast to $O(n^{2l})$ for existing methods. We conduct numerical experiments to compute the Bernstein coefficients for eleven polynomial problems (with dimensions varying from three to seven) defined over a unit box domain as well as a general box domain, with the existing methods and the proposed Matrix method. A comparison of the results shows the proposed algorithm to yield significant reductions in computational time for 'larger' number of Bernstein coefficients.

Keywords: Polynomial optimization, Bernstein coefficient, Interval analysis, Matrix operations
AMS subject classifications: 65-00

1 Introduction

Knowledge of the range of a multivariate polynomial is relevant for numerous investigations and applications in numerical and functional analysis, combinatorial optimization, and finite geometry. If a polynomial is written in the Bernstein basis over a box, then the range of the polynomial is bounded by the values of the minimum and maximum Bernstein coefficients [8].

*Submitted: July 27, 2012; Accepted: November 20, 2012.

Polynomial optimization using the Bernstein approach needs transformation of the given multivariate polynomial from its power form into its Bernstein form, and subsequently computation of the Bernstein coefficients. For obtaining these transformations, the Conventional method [6] has the disadvantage of high computational complexity for large dimensional problems. Although Garloff's method [3] is superior to the Conventional method, but on a general box-like domain its computational complexity is high when the number of Bernstein coefficients is large. Moreover, a study of Garloff's method reveals that it cannot be fully implemented in terms of only vector or array operations. Thus, the benefits of speedy vector-array operations (available in most programming languages) are not fully exploited in this method. Berchtold *et al.* [1] give an alternate method for converting univariate, bivariate and trivariate power-form polynomials into the equivalent Bernstein form. However, the method is difficult to generalize for dimensions higher than three. The method of computation of generalized Bernstein coefficients as given in [9, 12] is almost the same as that of the Conventional method. The scaled Bernstein coefficients approach of Sánchez-Reyes [10] reintroduces the binomial coefficients while computing the Bernstein coefficients, hence is inefficient for directly finding the latter. The parallel scheme of Garczarczyk [2] is specific to *parallel* processors, and is thus not appropriate for normal processing. Smith's method [11] of computation of the Bernstein coefficients of multivariate polynomials is efficient when the polynomials are sparse, i.e., when the number of terms of the polynomial is much less than the number of Bernstein coefficients.

In this paper, we present a new generalized technique, called the Matrix method, for computation of the Bernstein coefficients of multivariate polynomials. Using the proposed technique, the aforesaid limitations are reduced to a considerable extent. The proposed method is based fully on *matrix* operations, which enables speeding up of all the computations. In the proposed method, the given polynomial coefficients (irrespective of the dimensionality), are always arranged in the form of a *matrix* instead of a multidimensional array. The size of this matrix depends on the number of variables of the polynomial, and the maximum power of each variable in the polynomial. The computation of Bernstein coefficients then proceeds using only matrix operations such as inverse, multiplication, transpose and reshape. All these matrix operations are efficiently implemented in many popular programming languages as standard library routines, thus leading to much faster computations. Moreover, the transformation of the polynomial coefficients from a general box-like domain to a unit box domain is an inherent part of the proposed method.

The organization of the paper is as follows : in the next section, we give the notations and definitions of Bernstein polynomials and discuss about the basis conversion from power form to Bernstein form. In Section 3, we describe the existing methods for computation of Bernstein coefficients and also give the computational complexity associated with their algorithms. In Section 4, we present the proposed method for a bivariate case. The proposed Matrix method uses the *inverses* of three types of matrices. In the same section, initially we formulate a simple method to *directly* compute the inverses of the three required matrices. We also give the algorithms and the computational complexities of each formulation. Subsequently, we extend the same idea based on *matrix* operations to higher multivariate cases. In Section 5, we present the proposed algorithm for Bernstein coefficient computation, and in Section 6, we compare its computational complexity with those of the existing methods. In Section 7, we investigate the performances of all the three methods for Bernstein coefficients computation on eleven polynomial test problems with dimensions varying from three to seven taken from the literature. We present the conclusions in the last section followed by the description of the test problems.

2 Bernstein Forms

2.1 Notation and Definitions

Following the notations in [5], let $l \in \mathbb{N}$ be the number of variables and $x = (x_1, x_2, ..., x_l) \in \mathbb{R}^l$. Define a multi-index I as $I = (i_1, i_2, ..., i_l) \in \mathbb{N}^l$ and multi-power x as $x^I = (x_1^{i_1}, x_2^{i_2}, ..., x_l^{i_l})$. Further, define a multi-index of maximum degrees N as $N = (n_1, n_2, ..., n_l)$ and $I = (i_1, i_2, ..., i_l)$. Inequalities $I \leq N$ for multi-indices are meant component-wise, where $0 \leq i_k \leq n_k$, $k = 1, 2, ..., l$. Also, write $\binom{N}{I}$ for $\binom{n_1}{i_1}, ..., \binom{n_l}{i_l}$.

Let $\mathbf{x} = [\underline{x}, \overline{x}]$, $\overline{x} \geq \underline{x}$ be a real interval, where $\underline{x}$=inf $\mathbf{x}$ is the infimum, and $\overline{x}$=sup $\mathbf{x}$ is the supremum of the interval $\mathbf{x}$. The width of the interval $\mathbf{x}$ is defined as wid $\mathbf{x} = \overline{x} - \underline{x}$. For an l-dimensional interval vector or box $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_l)$, the width of $\mathbf{x}$ is wid $\mathbf{x}$ = (wid $\mathbf{x}_1$,wid $\mathbf{x}_2$, ...,wid $\mathbf{x}_l$).

We can write an l-variate polynomial p of degree N in the power form as

$$p(x) = \sum_{I \leq N} a_I x^I, a_I \in \mathbb{R}, \; x = (x_1, x_2, ..., x_l) \in \mathbb{R}^l \tag{1}$$

We can expand the multivariate polynomial in (1) into Bernstein polynomials over the l-dimensional box $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_l})$. Without loss of generality, we consider the unit box $\mathbf{u} = [0, 1]^l$, since any nonempty box $\mathbf{x}$ of $\mathbb{R}^l$ can be affinely mapped onto $\mathbf{u}$.

The transformation of a polynomial from its power form (1) into its Bernstein form results in

$$p(x) = \sum_{I \leq N} b_I(\mathbf{u}) B_{N,I}(x), \; x \in \mathbf{u} \tag{2}$$

The coefficients $b_I(\mathbf{u})$ are called the Bernstein coefficients of p over $\mathbf{u}$, and $B_{N,I}(x)$ is called the I^{th} Bernstein polynomial of degree N defined as

$$B_{N,I}(x) = B_{i_1}^{n_1}(x_1) B_{i_2}^{n_2}(x_2) ... B_{i_l}^{n_l}(x_l)$$

where,

$$B_{i_j}^{n_j}(x_j) = \binom{n_j}{i_j} x_j^{i_j} (1 - x_j)^{n_j - i_j}, \; i_j = 0, ..., n_j, \; j = 1, ..., l$$

Each set of coefficients (a_I or b_I) in (1) and (2) can be computed from the other as [1] :

$$a_I = \sum_{J \leq I} (-1)^{I-J} \binom{N}{I} \binom{I}{J} b_J$$

$$b_I(\mathbf{u}) = \sum_{J \leq I} \frac{\binom{I}{J}}{\binom{N}{J}} a_J, \; I \leq N \tag{3}$$

2.2 The Basis Conversion [1]

Most systems use the power form representation of a polynomial. When the Bernstein form is used, a conversion between the two bases is often necessary. For instance, in the univariate case, the equivalent power and Bernstein forms are

$$p(x) = \sum_{i=0}^{n} a_i x^i = \sum_{i=0}^{n} b_i^n B_i^n(x), \; x \in \mathbf{u} \tag{4}$$

In this subsection, we review the conversion of a *univariate* polynomial from the power form to its Bernstein form.. The polynomial $p(x)$ can be written in the following two ways

$$p(x) = \sum_{i=0}^{n} a_i x^i = \begin{pmatrix} 1 & x & x^2 & .. & .. & x^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ \vdots \\ a_n \end{pmatrix} = XA \tag{5}$$

or as

$$p(x) = \sum_{i=0}^{n} b_i^n B_i^n(x) = \begin{pmatrix} B_0^n(x) & B_1^n(x) & .. & .. & .. & B_n^n(x) \end{pmatrix} \begin{pmatrix} b_0^n \\ b_1^n \\ \vdots \\ \vdots \\ b_n^n \end{pmatrix} =: B_x B \tag{6}$$

Thus,

$$p(x) = XA = B_x B \tag{7}$$

If

$$U_x = \begin{pmatrix} \binom{n}{0}\binom{n}{0}(-1)^0 & 0 & ... & 0 \\ \binom{n}{0}\binom{n}{1}(-1)^1 & \binom{n}{1}\binom{n-1}{0}(-1)^0 & ... & 0 \\ \vdots & \vdots & ... & \vdots \\ \vdots & \vdots & ... & \vdots \\ \binom{n}{0}\binom{n}{n}(-1)^n & \binom{n}{1}\binom{n-1}{n-1}(-1)^{n-1} & ... & \binom{n}{n}\binom{n-n}{n-n}(-1)^{n-n} \end{pmatrix} \tag{8}$$

then, for a unit interval domain (see [1])

$$B_x B = X U_x B$$

or

$$B = U_x^{-1} A \tag{9}$$

where in the above, B_x is the Bernstein basis matrix, B is the Bernstein coefficient matrix, A is the polynomial coefficient matrix, and U_x is a lower triangular matrix given by (8).

On a general interval domain $[\underline{x}, \overline{x}]$, the Bernstein polynomials of degree $n \in \mathbb{N}$ are defined by

$$B_i^n(x) = \binom{n}{i} \frac{(x-\underline{x})^i (\overline{x}-x)^{n-i}}{(\overline{x}-\underline{x})^n}, i = 0, 1, ..., n \tag{10}$$

If

$$V_x = \begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 0 & \frac{1}{(\overline{x}-\underline{x})} & 0 & ... & 0 \\ \vdots & \vdots & \frac{1}{(\overline{x}-\underline{x})^2} & ... & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & .... & \frac{1}{(\overline{x}-\underline{x})^n} \end{pmatrix} \tag{11}$$

and

$$W_x = \begin{pmatrix} 1 & \binom{1}{0}(-\underline{x})^1 & \binom{2}{0}(-\underline{x})^2 & ... & \binom{n}{0}(-\underline{x})^n \\ 0 & \binom{1}{1}(-\underline{x})^{1-1} & \binom{2}{1}(-\underline{x})^{2-1} & ... & \binom{n}{1}(-\underline{x})^{n-1} \\ 0 & 0 & \binom{2}{2}(-\underline{x})^{2-2} & ... & \binom{n}{2}(-\underline{x})^{n-2} \\ \vdots & \vdots & 0 & ... & \vdots \\ \vdots & \vdots & \vdots & ... & \vdots \\ 0 & 0 & 0 & ... & \binom{n}{n}(-\underline{x})^{n-n} \end{pmatrix} \tag{12}$$

then, from the derivation given in [1]

$$XA = B_x\ B = XW_xV_xU_xB$$

or

$$B = U_x^{-1}V_x^{-1}W_x^{-1}A$$

where, A is the matrix containing the coefficients of the polynomial defined on a given general interval domain, V_x is a diagonal matrix given by (11), and W_x is an upper triangular matrix given by (12).

3 Existing Methods for Bernstein Coefficient Computation

In this section, we discuss three existing methods of computing the Bernstein coefficients along with their computational complexities.

3.1 Conventional Method

Let p be a polynomial on a general interval domain $\mathbf{x}$. Then,

$$p(x) = \sum_{I \le N} a_I x^I\ ,\ a_I \in \mathbb{R},\ x \in \mathbf{x}$$

In order to evaluate the Bernstein coefficients, the polynomial is first transformed onto a unit interval domain $\mathbf{u}$. The transformed polynomial [3] with the coefficients a'_I is given by

$$p(x) = \sum_{I \le N} a'_I x^I,\ a'_I \in \mathbb{R},\ x = (x_1, x_2, ..., x_l) \in \mathbf{u}$$

The Bernstein coefficients of p over $\mathbf{u}$ are then given by

$$b_I(\mathbf{u}) = \sum_{J \le I} \frac{\binom{I}{J}}{\binom{N}{J}} a'_J\ ,\ I \le N \tag{13}$$

The transformed polynomial coefficients are given by

$$a'_I = (\text{wid } \mathbf{x})^I \sum_{J=I}^{N} \binom{J}{I} (\inf \mathbf{x})^{J-I} a_J,\ I \le N \tag{14}$$

In the above, we need the binomial coefficients $C(J, I)$ defined as

$$C(J, I) := \binom{J}{I} \tag{15}$$

which can be computed using the following algorithm.

```
**Algorithm Binomial_coefficient** : $C(0:n, 0:n)$ = Binomial_coefficient $(n)$
Inputs : Degree $n$ of the polynomial.
Output : The binomial coefficients $C(0:n, 0:n)$.
BEGIN Algorithm
```

```
1. {Compute binomial coefficients $C(i,k)$}
   $C(1,0) = 1$,
   for $i = 2, ..., n$ do
      1. $C(i-1, i-1) = 1$, $C(i,0) = 1$
      2. for $k = 1, ..., (i-1)$ do $C(i,k) = \frac{i}{i-k} C((i-1),k)$.
2. {Return}
   return $C$.
END algorithm
```

Complexity of Algorithm Binomial_coefficient

- Step 1.2 requires 2 additions and multiplications, and is executed $(i-1)$ times. Therefore, it requires $2(i-1)$ number of additions and multiplications. Since i varies from 2 to n, step 1 requires a total number of

$$n(n-1) \tag{16}$$

additions and multiplications.

We can now implement (14), using the multivariate Horner scheme, and find the transformed polynomial coefficients a'_I.

3.1.1 Multivariate Horner Scheme [7, 13]

A multivariate polynomial in l variables

$$p(x) = \sum_{I=0}^{N} a_I x^I$$

may be represented in sum of products representation as

$$p(x_1, x_2, ..., x_l) = \sum_{I=0}^{N} a_I \prod_{r=1}^{l} x_r^{i_r} \tag{17}$$

Initially, set $p^{(1)}(x_1, x_2, ..., x_l) = p(x_1, x_2, ..., x_l)$ and expand it in powers of x_1 as

$$p^{(1)}(x_1, x_2, ..., x_l) = \sum_{i=0}^{n_1} x_1^i p_i^{(2)}(x_2, x_3, ..., x_l)$$

Using the Horner scheme, the above expression can be represented as

$$p^{(1)}(x_1, x_2, ..., x_l) = p_0^{(2)}(x_2, x_3, ..., x_l) + x_1(p_1^{(2)}(x_2, x_3, ..., x_l) + x_1(... + x_1 p_{n_1}^{(2)}(x_2, x_3, ..., x_l)))$$

Note that the coefficients $p_i^{(2)}$ are still polynomials with x_1 variable eliminated. The same expression can now be applied to the $(n_1 + 1)$ coefficients $p_i^{(2)}(x_2, x_3, ..., x_l)$ with x_2 as the indeterminate.

Table 1: Number of additions and multiplications required at each step to evaluate a polynomial using multivariate Horner scheme

Step	Additions	Multiplications
1	n	n
2	$n(n+1)$	$n(n+1)$
$\vdots$	$\vdots$	$\vdots$
l	$n(n+1)^{l-1}$	$n(n+1)^{l-1}$

We recursively expand the coefficients in the same way until polynomials in only x_l are obtained. The expansion is repeated l times and in each step we obtain coefficients with one variable less. Hence,

$$p^{(r)}(x_r, x_{r+1}, ..., x_l) = \sum_{i=0}^{n_r} x_r^i p^{(r+1)}(x_{r+1}, x_{r+2}, ..., x_l),\ r = 1, 2, ..., l$$

For the sake of convenience, we assume here all powers n_r equal to n. The additions and multiplications required at each step are shown in Table 1.

From Table 1, the total additions and multiplications is obtained as

$$\sum_{r=1}^{l} n(n+1)^{r-1} = (n+1)^l - 1$$

leading to the following theorem.

Theorem 3.1 *[13] The evaluation of an l-variate polynomial in power form of maximum degree n in all variables requires at most $(n+1)^l - 1$ additions and $(n+1)^l - 1$ multiplications.*

The Horner scheme as applied to (14) is essentially the same as applied to (17), except that instead of the indeterminates x_r, we have the products of the binomial coefficients and the infima of the box.

For the sake of illustration, consider the multivariate Horner scheme as applied to the transformation of a *bivariate* polynomial from a general box to a unit box domain. For a bivariate polynomial, the transformed polynomial coefficients are given by

$$a'_{i_1 i_2} = (\text{wid}\ \mathbf{x_1})^{i_1} (\text{wid}\ \mathbf{x_2})^{i_2} \sum_{j_1=i_1}^{n_1} \binom{j_1}{i_1} \inf \mathbf{x_1}^{j_1-i_1} \sum_{j_2=i_2}^{n_2} \binom{j_2}{i_2} \inf \mathbf{x_2}^{j_2-i_2} a_{j_1 j_2}$$

where $0 \leq i_{1,2} \leq n_{1,2}$ and the indeterminates are $\binom{j_1}{i_1} \inf \mathbf{x_1}^{j_1-i_1}$ and $\binom{j_2}{i_2} \inf \mathbf{x_2}^{j_2-i_2}$. By the application of Horner's method, the following parenthesizing is obtained :

$$\begin{aligned} a'_{i_1 i_2} \quad = \quad & (\text{wid}\ \mathbf{x_1})^{i_1} (\text{wid}\ \mathbf{x_2})^{i_2} \Bigg\{ a_{i_1 i_2} + ... + \binom{n_1 - 1}{i_1} \inf \mathbf{x_1} \{ a_{n_1 - 1 i_2} + ... \\ & + \binom{n_1}{i_1} \inf \mathbf{x_1} \{ a_{n_1 i_2} + ... + \binom{n_2 - 2}{i_2} \inf \mathbf{x_2} \{ a_{n_1 n_2 - 2} \\ & + \binom{n_2 - 1}{i_2} \inf \mathbf{x_2} \ \Bigg\{ a_{n_1 n_2 - 1} + \binom{n_2}{i_2} \inf \mathbf{x_2} a_{n_1 n_2} \Bigg\} \Bigg\} \Bigg\} \Bigg\} \Bigg\} \end{aligned}$$

Based on the above multivariate Horner scheme, we present here an algorithm for transformation of the coefficients of an l-variate polynomial from a general box to a unit box domain. We assume here for simplicity that degree N of all the variables are equal to n. The algorithm initially computes all the powers (from 1 to n) of the widths of all the sides of a general box, and then finds all the coefficients of the transformed polynomial using (14).

Algorithm Transformation_Horner : $a_I^{'}$ = Transformation $(N,\ \mathbf{x},\ a_I, C(0:n, 0:n))$

Inputs : Degree N of the polynomial (which is taken to be n), the l-dimensional box $\mathbf{x}$, all the polynomial coefficients $a_I = a(i_1, ...i_l)$, and the binomial coefficients $C(0:n, 0:n)$.

Output : Coefficients $a_I^{'} = a^{'}(i_1, ...i_l)$ of the transformed polynomial.

```
BEGIN Algorithm
1. {Compute edge lengths of $\mathbf{x}$, i.e., widths of the box $\mathbf{x}$}
   for $r = 1, ..., l$ do wid $\mathbf{x}_r = (\overline{x}_r - \underline{x}_r)$
2. {All the powers of the widths}
   for $r = 1, .., l$ do wid $\mathbf{x}_r^0 = 1$
       1. for $i_r = 1, ..., n$ do wid $\mathbf{x}_r^{i_r} = \left(\text{wid } \mathbf{x}_r^{(i_r-1)}\right)(\text{wid } \mathbf{x}_r)$
3. {Coefficients of the transformed polynomial}
   for $i_1 = 0, ..., n$
       ⋮
     for $i_r = 0, ..., n$
         ⋮
       1. for $i_l = 0, ..., n$ do $d(n+1, i_2, ..., i_l) = 0$
           1. for $j_1 = n, ..., i_1$ do $d(j_1, n+1, i_3, ..., i_l) = 0$
                  ⋮
              for $j_r = n, ..., i_r$ do $d(j_1, ..., j_r, n+1, ..., i_l) = 0$
                  ⋮
               for $j_l = n, ..., i_l$
               $dd(j_1, ..., j_l) = a(j_1, ..., j_l) + C(j_l+1, i_l)\inf \mathbf{x}_l d(j_1, ..., j_l+1)$.
                  ⋮
              $d(j_1, ..., j_r, i_{r+1}, ..., i_l) = d(j_1, ..., j_r, ..., i_l) + C(j_r+1, i_r)\inf \mathbf{x}_r d(j_1, ...j_r+1, ..., i_l)$.
                ⋮
              $d(j_1, i_2, ..., i_l) = d(j_1, i_2, ..., i_l) + C(j_1+1, i_1)\inf \mathbf{x}_1 d(j_1+1, i_2, ..., i_l)$.
           2. $a^{'}(i_1, ...i_l) = d(i_1, ..., i_l)$wid $\mathbf{x}_1^{i_1}$, ...,wid $\mathbf{x}_l^{i_l}$
4. {Return}
   return $a_I^{'}$.
```

END Algorithm.

Complexity of Algorithm Transformation_Horner

The computational complexity of the algorithm for transformation of the l-variate polynomial is given below.

- Step 1 requires l additions.
- Step 2.1 requires n multiplications and is executed l times. Therefore, step 2 requires ln multiplications.
- Step 3.1.1.1 requires $(n - i_l + 1)$ additions. Hence step 3.1.1 requires
$$\prod_{r=1}^{l} (n - i_r + 1) - 1 \tag{18}$$
additions. Since i_r varies from 0 to n, step 3 requires
$$\left(\left(\frac{(n+1)(n+2)}{2} \right)^l - (n+1)^l \right) \tag{19}$$
additions.
- Assuming that the binomial coefficients are precomputed, step 3.1.1.1 requires $2(n - i_r + 1)$ multiplications. Hence step 3.1.1 requires $2\left(\prod_{r=1}^{l} (n - i_r + 1) - 1 \right)$ multiplications. Step 3.1.2 requires l multiplications. Hence, step 3.1 requires
$$2\left(\prod_{r=1}^{l} (n - i_r + 1) - 1 \right) + l \tag{20}$$
multiplications. Since i_r varies from 0 to n, step 3 requires
$$2\left(\left(\frac{(n+1)(n+2)}{2} \right)^l - (n+1)^l \right) + l(n+1)^l \tag{21}$$
multiplications.
- Summarizing the above steps, for the l-variate case, the total additions required is
$$l + \left(\left(\frac{(n+1)(n+2)}{2} \right)^l - (n+1)^l \right) \tag{22}$$
while the total multiplications required is
$$ln + 2\left(\frac{(n+1)(n+2)}{2} \right)^l - 2(n+1)^l + l(n+1)^l \tag{23}$$

Remark 1 *For the l-variate case, the first transformed polynomial coefficient is obtained with $i_r = 0$ for $r = 1, ..., l$. Therefore, from (18) and (20) we see that $\left((n+1)^l - 1\right)$ additions and $\left(2\left((n+1)^l - 1\right) + l\right)$ multiplications are needed for the first transformed polynomial coefficient. Thus, for one term of the transformed polynomial the computational complexity of the transformation onto a unit box is $O(n^l)$.*

Remark 2 *From (22) and (23), the computational complexity of the transformation of all the terms of the polynomial onto a unit box is $O(n^{2l})$.*

3.1.2 Conventional Method

After transforming the polynomial coefficients from a given general box to a unit box, we compute the Bernstein coefficients using (13). We call this method as the *Conventional* method of computing the Bernstein coefficients.

We present below an algorithm Conventional_Bernstein_coefficient to compute the Bernstein coefficients using (13). We again assume here that degree N of all the variables are equal to n.

Algorithm Conventional_Bernstein_coefficient : $B(\mathbf{u}) =$ Conventional_Bernstein_coefficient $(N, \mathbf{x}, a_I)$

Inputs : Degree N of the polynomial (which is taken to be n), a l-dimensional box $\mathbf{x}$ and the coefficients a_I of the polynomial.

Output : Bernstein coefficients b_I on the unit box or patch $B(\mathbf{u})$ of the Bernstein coefficients.

```
BEGIN Algorithm

1. {Compute binomial coefficients}
   $C(0:n, 0:n)$ =Binomial_coefficient $(n)$
2. {Execute Algorithm Transformation_Horner}
   $a'_I$ =Transformation $(N, \mathbf{x}, a_I, C(0:n, 0:n))$
3. {Bernstein coefficients of the transformed polynomial}
   for $I = 0, ..., N$ do $b_I = 0$
      1. for $J = 0, ..., I$ do $b_I = b_I + a'_J C(I, J)/C(N, J)$.
4. {Return}
   return the patch $B(\mathbf{u}) = b_I(\mathbf{u})$.

END Algorithm.
```

Complexity of Algorithm Conventional_Bernstein_coefficient
The complexity of the algorithm is calculated for the l-variate case as follows.

- Complexity of step 1 has been found earlier in (16). From (16), the total additions and multiplications is
$$n(n-1)$$
- Complexity of step 2 has been found earlier in (22) and (23). From (22), the total additions is
$$l + \left(\left(\frac{(n+1)(n+2)}{2} \right)^l - (n+1)^l \right) \tag{24}$$
while from (23) the total multiplications is
$$ln + 2\left(\frac{(n+1)(n+2)}{2} \right)^l - 2(n+1)^l + l(n+1)^l \tag{25}$$

- Step 3.1 requires $\prod_{r=1}^{l}(i_r+1)$ additions. Since i_r takes values from 0 to n, step 3 requires
$$\left(\frac{(n+1)(n+2)}{2}\right)^l \tag{26}$$
additions.
- Step 3.1 requires $2l\prod_{r=1}^{l}(i_r+1)$ multiplications. Since i_r takes values from 0 to n, step 3 requires a total of
$$2l\left(\frac{(n+1)(n+2)}{2}\right)^l \tag{27}$$
multiplications.
- Neglecting the work required for the binomial coefficient computation in step 1, from (24) and (26) the total additions required is
$$l+\left(\left(\frac{(n+1)(n+2)}{2}\right)^l-(n+1)^l\right)+\left(\frac{(n+1)(n+2)}{2}\right)^l \tag{28}$$
while, from (25) and (27) the total multiplications required is
$$ln+2\left(\frac{(n+1)(n+2)}{2}\right)^l-2(n+1)^l+l(n+1)^l+2l\left(\frac{(n+1)(n+2)}{2}\right)^l \tag{29}$$
- For a unit box domain, neglecting again the work required for computing binomial coefficient computations, we have the following : since step 2 for domain transformation is not required, from (26) the total additions is
$$\left(\frac{(n+1)(n+2)}{2}\right)^l \tag{30}$$
while from (27) the total multiplications is
$$2l\left(\frac{(n+1)(n+2)}{2}\right)^l \tag{31}$$

Remark 3 *From (28) to (31), we see that the computational complexity of the Conventional method for a unit box domain as well as for a general box domain is $O(n^{2l})$.*

3.2 Generalized Bernstein Polynomial [9, 12]

In order to generalize (4) to a general interval domain $\mathbf{x}=[\underline{x},\overline{x}]$, we can write the polynomial in terms of the shifted polynomial coefficients c_i as
$$p(x)=\sum_{i=0}^{n}a_ix^i=\sum_{i=0}^{n}c_i(x-\underline{x})^i=\sum_{i=0}^{n}b_i^nB_i^n(x),\ x\in\mathbf{x} \tag{32}$$
where $B_i^n(x)$ is the generalized Bernstein polynomial of degree n and is defined as (10) and b_i^n is the generalized Bernstein coefficient and is defined as
$$b_i^n=\sum_{j=0}^{i}c_j(\overline{x}-\underline{x})^i\frac{\binom{i}{j}}{\binom{n}{j}},\ j=0,1,...,n$$

From (32) c_i is obtained as

$$c_i = \sum_{j=i}^{n} \binom{j}{i} \underline{x}^{j-i} a_j$$

Following the computational complexity calculations given in Section 3.1, to compute the coefficients of the shifted polynomial c_i the additions required is found to be $\left(\left(\frac{(n+1)(n+2)}{2}\right)^l - (n+1)^l\right)$, and the multiplications required is $2\left(\left(\frac{(n+1)(n+2)}{2}\right)^l - (n+1)^l\right)$.

The additions required to compute the generalized Bernstein coefficients is $\left(\frac{(n+1)(n+2)}{2}\right)^l$ and the multiplications is $3l\left(\frac{(n+1)(n+2)}{2}\right)^l$. Hence, the total additions and multiplications required to compute of the generalized Bernstein coefficients (including the computation of the coefficients of the shifted polynomial) is respectively

$$l + \left(\left(\frac{(n+1)(n+2)}{2}\right)^l - (n+1)^l\right) + \left(\frac{(n+1)(n+2)}{2}\right)^l \tag{33}$$

and

$$ln + 2\left(\frac{(n+1)(n+2)}{2}\right)^l - 2(n+1)^l + 3l\left(\frac{(n+1)(n+2)}{2}\right)^l \tag{34}$$

Remark 4 *From (33) and (34), we see that the computational complexity of the generalized Bernstein coefficients is $O(n^{2l})$. Since its order of computational complexity is the same as that of the Conventional method, we shall not consider it for further comparison.*

3.3 Garloff's Method

Garloff [3] proposed a method wherein the Bernstein coefficients of a polynomial p over the unit box domain $\mathbf{u}$ are computed by a difference table scheme. The scheme reduces the number of the products appearing in the Conventional method given by (13). This method is more efficient than the Conventional method, as it requires fewer arithmetic operations and computation of a smaller number of binomial coefficients. The method is explained in detail in [3] for the bivariate case, and can be readily generalized to higher multivariate cases.

First, the polynomial defined on a general box is transformed onto a unit box using (14). This transformation is done using the multivariate Horner scheme (refer to Section 3.1). After the transformation of the polynomial coefficients from a given general box domain to a unit box domain is done, the Bernstein coefficients are computed.

In brief, the Bernstein coefficients are computed in two steps. First, compute

$$\Delta_K b_0 = \binom{N}{K}^{-1} a_K,\ K \leq N \tag{35}$$

where, Δ_K is an l-dimensional forward difference operator defined by

$$\Delta_K b_I := \sum_{J \leq K} (-1)^{K-J} \binom{K}{J} b_{I+J},\ K \leq N - I$$

and a_I are the coefficients of the polynomial defined on a unit box. Then, compute the Bernstein coefficients from (35), using the recurrence relations

$$\Delta_0 b_I = b_I$$

$$\Delta_I b_I + \Delta_{I_{r,1}} b_I = \Delta_I b_{I_{r,1}}$$

Details of the method are given in [3].

Complexity of Garloff's method

As reported in [3, 4], Garloff's method on a unit box domain requires

$$l\frac{n(n+1)^l}{2} \tag{36}$$

additions, and

$$l(n+1)^l \tag{37}$$

multiplications.

For a general box domain, we also need to consider the work required for transformation of the polynomial coefficients on to a unit box. Using (22) and (23), and without taking into considerations the work done in computing the binomial coefficients, on a general box domain, the total additions required is

$$l + \left(\left(\frac{(n+1)(n+2)}{2}\right)^l - (n+1)^l\right) + l\frac{n(n+1)^l}{2} \tag{38}$$

while, the total multiplications required is

$$ln + 2\left(\frac{(n+1)(n+2)}{2}\right)^l - 2(n+1)^l + 2l(n+1)^l \tag{39}$$

Remark 5 *Thus, from (36) and (37), we conclude that the computational complexity of Garloff's method for a unit box domain is $O(n^{l+1})$, and from (38) and (39), the computational complexity for a general box domain (including the work required for the multivariate Horner scheme) is $O(n^{2l})$.*

4 The Proposed Matrix Method

The Bernstein coefficient matrix B for a general domain for the univariate case is given by

$$B = U_x^{-1} V_x^{-1} W_x^{-1} A = MA \tag{40}$$

where the polynomial coefficients matrix $A = \begin{bmatrix} a_0 & a_1 & : & : & a_n \end{bmatrix}^T$, the superscript T denotes 'transpose' and M is the product matrix given by

$$M := U_x^{-1} V_x^{-1} W_x^{-1} = \widetilde{U}_x \widetilde{V}_x \widetilde{W}_x \tag{41}$$

with $\widetilde{U}_x := U_x^{-1}$, $\widetilde{V}_x := V_x^{-1}$ and $\widetilde{W}_x := W_x^{-1}$, where U_x, V_x and W_x are given as in (8), (11) and (12) respectively.

The proposed method uses the *inverses* of matrices U_x, V_x and W_x. As matrix inversion is costly, we formulate a simple method to *directly* compute the inverses of the three required matrices, thus reducing the computational burden of the proposed method. The inverse of U_x is given by

$$\widetilde{U}_x = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & \binom{1}{0}\binom{n}{1}^{-1} & 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & \binom{2}{1}\binom{n}{1}^{-1} & \binom{2}{0}\binom{n}{2}^{-1} & 0 & 0 & 0 & \dots & 0 \\ 1 & \binom{3}{2}\binom{n}{1}^{-1} & \binom{3}{1}\binom{n}{2}^{-1} & \binom{3}{0}\binom{n}{3}^{-1} & 0 & 0 & \dots & 0 \\ 1 & \binom{4}{3}\binom{n}{1}^{-1} & \binom{4}{2}\binom{n}{2}^{-1} & \binom{4}{1}\binom{n}{3}^{-1} & \binom{4}{0}\binom{n}{4}^{-1} & 0 & \dots & 0 \\ 1 & \binom{5}{4}\binom{n}{1}^{-1} & \binom{5}{3}\binom{n}{2}^{-1} & \binom{5}{2}\binom{n}{3}^{-1} & \binom{5}{1}\binom{n}{4}^{-1} & \binom{5}{0}\binom{n}{5}^{-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 \end{pmatrix} \tag{42}$$

The inverse of V_x is given by

$$\widetilde{V}_x = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & (\overline{x}-\underline{x}) & 0 & 0 & \dots & 0 \\ 0 & 0 & (\overline{x}-\underline{x})^2 & 0 & \dots & 0 \\ 0 & 0 & 0 & (\overline{x}-\underline{x})^3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & (\overline{x}-\underline{x})^n \end{pmatrix} \tag{43}$$

and the inverse of W_x is given by

$$\widetilde{W}_x = \begin{pmatrix} 1 & \binom{1}{0}(\underline{x})^1 & \binom{2}{0}(\underline{x})^2 & \binom{3}{0}(\underline{x})^3 & \dots & \binom{n}{0}(\underline{x})^n \\ 0 & 1 & \binom{2}{1}(\underline{x})^1 & \binom{3}{1}(\underline{x})^2 & \dots & \binom{n}{1}(\underline{x})^{n-1} \\ 0 & 0 & 1 & \binom{3}{2}(\underline{x})^1 & \dots & \binom{n}{2}(\underline{x})^{n-2} \\ 0 & 0 & 0 & 1 & \dots & \binom{n}{3}(\underline{x})^{n-3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix} \tag{44}$$

Based on (41) to (44), we next give algorithms for the computations of the inverse matrices $\widetilde{U}_x$, $\widetilde{V}_x$ and $\widetilde{W}_x$, and the product matrix M. Using Algorithm Binomial_coefficient given earlier in Section 3.1, we first precompute the binomial coefficients and use them as inputs in the algorithms for computing $\widetilde{U}_x$ using (42) and $\widetilde{W}_x$ using (44).

```
**Algorithm InverseUx** : $\widetilde{U}_x(0:n,0:n) = \text{InverseUx}\ (n,\ C(0:n,0:n))$
Inputs : Degree $n$ of the polynomial, and the binomial coefficients $C(0:n,0:n)$.
Output : The matrix $\widetilde{U}_x := U_x^{-1}$
BEGIN Algorithm
1. {Construction of inverse of matrix $U_x$}
   $\widetilde{U}_x(0:n,0)=1$, $\widetilde{U}_x(0,1:n)=0$, $\widetilde{U}_x(n,1:n)=1$,
2. for $i=1,...,(n-1)$ do
    1. for $j=1,...,i$ do $\widetilde{U}_x(i,j)=C(i,i-j)/C(n,j)$
    2. $\widetilde{U}_x(i,i+1:n)=0$.
3. {Return}
   return $\widetilde{U}_x$.
```

END algorithm

Complexity of Algorithm InverseUx

- Step 2.1 requires i multiplications. Since i takes values from 1 to $n-1$, step 2 requires
$$\frac{n(n-1)}{2} \tag{45}$$
multiplications.
- No additions are involved here.

Algorithm InverseVx : $\widetilde{V}_x(0:n,0:n) = \text{InverseVx}\ (n,\ \mathbf{x})$
Inputs : Degree n of the polynomial, and the domain box $\mathbf{x} = [\underline{x}, \overline{x}]$.
Output : The matrix $\widetilde{V}_x := V_x^{-1}$
BEGIN Algorithm

```
1. {Compute edge length of $\mathbf{x}$, i.e., width of the box}
   wid $\mathbf{x} = (\overline{x} - \underline{x})$
2. {Construction of inverse of matrix $V_x$}
   $\widetilde{V}_x(0:n,0:n) = 0$, $\widetilde{V}_x(0,0) = 1$, wid $\mathbf{x}^0 = 1$,
   for $i = 1,...,n$ do
      wid $\mathbf{x}^i = \left(\text{wid } \mathbf{x}^{(i-1)}\right)(\text{wid } \mathbf{x})$
      $\widetilde{V}_x(i,i) =$wid $\mathbf{x}^i$.
3. {Return}
   return $\widetilde{V}_x$.
```

END algorithm

Complexity of Algorithm InverseVx

- Step 1 requires only one addition.
- Step 2 requires 1 multiplication and is executed n times, therefore it requires n multiplications.

Algorithm InverseWx : $\widetilde{W}_x(0:n,0:n) = \text{InverseWx}\ (n,\ \mathbf{x},\ C(0:n,0:n))$
Inputs : Degree n of the polynomial, the infimum inf $\mathbf{x}$ of the box $\mathbf{x}$, and the binomial coefficients $C(0:n,0:n)$.
Output : The matrix $\widetilde{W}_x := W_x^{-1}$.
BEGIN Algorithm

```
1. {All the powers of the infimum}
   inf $\mathbf{x}^0 = 1$,
   for $i = 1,...,n$ do inf $\mathbf{x}^i = \left(\text{inf } \mathbf{x}^{(i-1)}\right)(\text{inf } \mathbf{x})$
2. {Construction of inverse of matrix $W_x$}
   $\widetilde{W}_x(0,0) = 1$,
   for $i = 0,...,n-1$ do
```

```
        1. for $j = i+1, ..., n$ do
              $\widetilde{W}_x(i,j) = C(j,i)\inf \mathbf{x}^{(j-i)}$.
        2. $\widetilde{W}_x(i+1, 0:i) = 0$, $\widetilde{W}_x(i+1, i+1) = 1$.
    3. {Return}
       return $\widetilde{W}_x$.
END algorithm
```

Complexity of Algorithm InverseWx

- Step 1 requires n multiplications.
- Step 2.1 requires 1 multiplication and it is executed $(n-i)$ times, where i is the index in step 2. Since i varies from 0 to $n-1$, it requires $n(n+1)/2$ multiplications.
- Therefore, from steps 1 and 2, the total multiplications is

$$n + \frac{n(n+1)}{2} = \frac{n(n+3)}{2} \tag{46}$$

- No additions are involved here.

Algorithm ProductM : $M(0:n, 0:n) = \text{ProductM}\ (\widetilde{U}_x, \widetilde{V}_x, \widetilde{W}_x)$
Inputs : The inverse matrices $\widetilde{U}_x$, $\widetilde{V}_x$, $\widetilde{W}_x$.
Output : The product matrix $M := \widetilde{U}_x$, $\widetilde{V}_x$, $\widetilde{W}_x$.
BEGIN Algorithm

```
1. {Product of all the inverse matrices}
   $M = \widetilde{U}_x \widetilde{V}_x \widetilde{W}_x$.
2. {Return}
   return $M$.
END algorithm
```

Complexity of Algorithm ProductM

- Since $\widetilde{V}_x$ is a diagonal matrix (with first element as unity) and $\widetilde{W}_x$ is an upper triangular matrix (with all its diagonal elements as unity), the product of these two would be an upper triangular matrix. So, there would be $n(n+1)/2$ multiplications with no additions.
- Since $\widetilde{U}_x$ is a lower triangular matrix (with all elements of first column as unity) and the product $\widetilde{V}_x\widetilde{W}_x$ is an upper triangular matrix (with first element as unity), the product of these two matrices requires $\sum_{i=1}^{n} i\,(2(n-i)+1)$ additions and $\sum_{i=1}^{n+1} i\,(2(n+1-i)+1)$ multiplications.
- Summing up, the total additions required is

$$\sum_{i=1}^{n} i\,(2(n-i)+1) \tag{47}$$

whereas, the total multiplications required is

$$\frac{n(n+1)}{2} + \sum_{i=1}^{n+1} i\,(2(n+1-i)+1) \tag{48}$$

Remark 6 *All the elements of first column of product matrix M are unity.*

4.1 Bivariate Case

In this section, we incorporate the results of the univariate case in [1] to devise an alternate way of computing the Bernstein coefficients of a bivariate polynomial using matrix multiplications. A bivariate polynomial in the power form can be expressed as

$$\begin{aligned} p(x_1,x_2) &= a_{00} + a_{10}x_1^1 + a_{20}x_1^2 + a_{01}x_2 + a_{11}x_1x_2 + \dots\dots + a_{n_1n_2}x_1^{n_1}x_2^{n_2} \\ &= X_1AX_2^T = X_2(X_1A)^T \end{aligned} \tag{49}$$

where

$$X_1 = \begin{pmatrix} 1 & x_1 & x_1^2 & .. & .. & x_1^{n_1} \end{pmatrix}$$
$$X_2 = \begin{pmatrix} 1 & x_2 & x_2^2 & .. & .. & x_2^{n_2} \end{pmatrix}$$

and

$$A = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n_2} \\ a_{10} & a_{11} & \dots & a_{1n_2} \\ : & : & : & : \\ : & : & : & : \\ a_{n_10} & a_{n_11} & \dots & a_{n_1n_2} \end{pmatrix}$$

The same polynomial can be expressed in the Bernstein form as

$$\begin{aligned} p(x_1,x_2) &= b_{00}B_0^{n_1}(x_1)B_0^{n_2}(x_2) + b_{10}B_1^{n_1}(x_1)B_0^{n_2}(x_2) + b_{20}B_2^{n_1}(x_1)B_0^{n_2}(x_2) \\ &\quad + b_{01}B_0^{n_1}(x_1)B_1^{n_2}(x_2) + \dots\dots + b_{n_1n_2}B_{n_1}^{n_1}(x_1)B_{n_2}^{n_2}(x_2) \\ &= B_{x_1}BB_{x_2}^T = B_{x_2}\left(B_{x_1}B\right)^T \end{aligned} \tag{50}$$

where

$$B_{x_1} = \begin{pmatrix} B_0^{n_1}(x_1) & B_1^{n_1}(x_1) & .. & .. & .. & B_{n_1}^{n_1}(x_1) \end{pmatrix}$$
$$B_{x_2} = \begin{pmatrix} B_0^{n_2}(x_2) & B_1^{n_2}(x_2) & .. & .. & .. & B_{n_2}^{n_2}(x_2) \end{pmatrix}$$

and

$$B = \begin{pmatrix} b_{00} & b_{01} & \dots & b_{0n_2} \\ b_{10} & b_{11} & \dots & b_{1n_2} \\ : & : & : & : \\ : & : & : & : \\ b_{n_10} & b_{n_11} & \dots & b_{n_1n_2} \end{pmatrix}$$

From (49) and (50)

$$B_{x_2}\left(B_{x_1}B\right)^T = X_2\left(X_1A\right)^T \tag{51}$$

where the superscript T denotes the 'transpose', which means changing the order of the indices of the elements of the two-dimensional array.

On a unit box domain, from (8)

$$B_{x_1} = X_1U_{x_1} \text{ and } B_{x_2} = X_2U_{x_2} \tag{52}$$

Substituting B_{x_1} and B_{x_2} from (52) in (51) gives

$$\begin{aligned}
X_2 U_{x_2} \left(X_1 U_{x_1} B\right)^T &= X_2 \left(X_1 A\right)^T \\
\Rightarrow U_{x_2} \left(X_1 U_{x_1} B\right)^T &= \left(X_1 A\right)^T \\
\Rightarrow X_1 U_{x_1} B \left(U_{x_2}\right)^T &= X_1 A \\
\Rightarrow B \left(U_{x_2}\right)^T &= U_{x_1}^{-1} A \\
\Rightarrow \left(B \left(U_{x_2}\right)^T\right)^T &= \left(U_{x_1}^{-1} A\right)^T \\
\Rightarrow U_{x_2} B^T &= \left(U_{x_1}^{-1} A\right)^T \\
\Rightarrow B^T &= U_{x_2}^{-1} \left(U_{x_1}^{-1} A\right)^T \\
\Rightarrow B &= \left(U_{x_2}^{-1} \left(U_{x_1}^{-1} A\right)^T\right)^T \qquad (53)
\end{aligned}$$

On a general box domain, from (12)

$$B_{x_1} = X_1 W_{x_1} V_{x_1} U_{x_1} \text{ and } B_{x_2} = X_2 W_{x_2} V_{x_2} U_{x_2} \qquad (54)$$

Substituting B_{x_1} and B_{x_2} from (54) in (51) gives

$$\begin{aligned}
X_2 W_{x_2} V_{x_2} U_{x_2} \left(X_1 W_{x_1} V_{x_1} U_{x_1} B\right)^T &= X_2 \left(X_1 A\right)^T \\
\Rightarrow W_{x_2} V_{x_2} U_{x_2} \left(X_1 W_{x_1} V_{x_1} U_{x_1} B\right)^T &= \left(X_1 A\right)^T \\
\Rightarrow X_1 W_{x_1} V_{x_1} U_{x_1} B \left(W_{x_2} V_{x_2} U_{x_2}\right)^T &= X_1 A \\
\Rightarrow B \left(W_{x_2} V_{x_2} U_{x_2}\right)^T &= U_{x_1}^{-1} V_{x_1}^{-1} W_{x_1}^{-1} A \\
\Rightarrow \left(B \left(W_{x_2} V_{x_2} U_{x_2}\right)^T\right)^T &= \left(U_{x_1}^{-1} V_{x_1}^{-1} W_{x_1}^{-1} A\right)^T \\
\Rightarrow W_{x_2} V_{x_2} U_{x_2} B^T &= \left(U_{x_1}^{-1} V_{x_1}^{-1} W_{x_1}^{-1} A\right)^T \\
\Rightarrow B^T &= U_{x_2}^{-1} V_{x_2}^{-1} W_{x_2}^{-1} \left(U_{x_1}^{-1} V_{x_1}^{-1} W_{x_1}^{-1} A\right)^T \\
\Rightarrow B &= \left(U_{x_2}^{-1} V_{x_2}^{-1} W_{x_2}^{-1} \left(U_{x_1}^{-1} V_{x_1}^{-1} W_{x_1}^{-1} A\right)^T\right)^T
\end{aligned}$$

Define

$$M_1 := U_{x_1}^{-1} V_{x_1}^{-1} W_{x_1}^{-1} \qquad (55)$$

$$M_2 := U_{x_2}^{-1} V_{x_2}^{-1} W_{x_2}^{-1} \qquad (56)$$

then

$$B = \left(M_2 \left(M_1 A\right)^T\right)^T \qquad (57)$$

Remark 7 *All the elements of the first column of product matrices M_1 and M_2 respectively given by (55) and (56) are unity.*

4.2 Higher Multivariate Cases

In order to readily generalize the procedure to higher multivariate cases, we first derive the formula for computing the Bernstein coefficients for the trivariate case, on lines similar to the bivariate case developed in Section 4.1. We shall then extend the same to the general multivariate case.

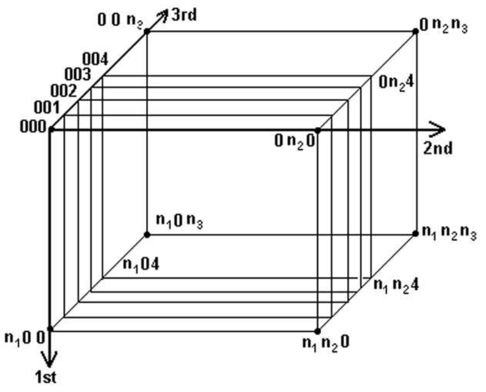

Figure 1: Three-dimensional coefficient array, showing the coefficients of the third coordinate direction placed on the parallel faces

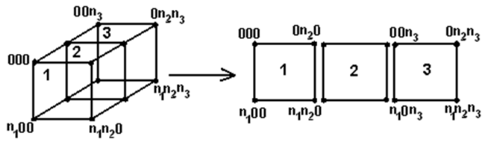

Figure 2: Two-dimensional or matrix representation of a three-dimensional coefficient array, showing the vertices

A trivariate polynomial in power form can be expressed as

$$\begin{aligned} p(x_1, x_2, x_3) &= a_{000} + a_{100}x_1^1 + a_{010}x_2 + a_{110}x_1x_2 + a_{001}x_3 + ... \\ &\quad + a_{n_1n_2n_3}x_1^{n_1}x_2^{n_2}x_3^{n_3} \\ &= \sum_{i_1=0}^{n_1}\sum_{i_2=0}^{n_2}\sum_{i_3=0}^{n_3} a_{i_1i_2i_3}x_1^{i_1}x_2^{i_2}x_3^{i_3} = \sum_{I\le N} a_I x^I \end{aligned} \tag{58}$$

where, a_I is a three-dimensional coefficient array shown in Figure 1, with n_1, n_2 and n_3 as the respective degrees of the variables x_1, x_2 and x_3.

We can view the three dimensional array a_I as a matrix A having $n_1 + 1$ rows and $(n_2 + 1)(n_3 + 1)$ columns arranged as shown in Figure 2.

Then, the polynomial p in (58) can be written by means of matrix multiplication as

$$p(x_1, x_2, x_3) = X_3 \left(X_2 \left(X_1 A \right)^T \right)^T \tag{59}$$

where, X_1, X_2, and X_3 are vectors, and A is a three-dimensional coefficient array represented in the form of a matrix. Similarly, if B_{x_1}, B_{x_2} and B_{x_3} are the Bernstein vectors in the variables x_1, x_2 and x_3 respectively, then in the Bernstein form, the polynomial $p(x_1, x_2, x_3)$ is

$$p(x_1, x_2, x_3) = B_{x_3} \left(B_{x_2} \left(B_{x_1} B \right)^T \right)^T \tag{60}$$

Note that B is also represented in the form of a matrix. Equating (59) and (60) and thereafter using the properties of matrix operations, on a unit box domain we can obtain B as

$$B = \left(U_{x_3}^{-1} \left(U_{x_2}^{-1} \left(U_{x_1}^{-1} A \right)^T \right)^T \right)^T \tag{61}$$

Similarly, on a general box domain we can obtain B as

$$B = \left(U_{x_3}^{-1} V_{x_3}^{-1} W_{x_3}^{-1} \left(U_{x_2}^{-1} V_{x_2}^{-1} W_{x_2}^{-1} \left(U_{x_1}^{-1} V_{x_1}^{-1} W_{x_1}^{-1} A \right)^T \right)^T \right)^T$$

and write it as

$$B = \left(M_3 \left(M_2 \left(M_1 A \right)^T \right)^T \right)^T \tag{62}$$

where the matrices M_1, M_2, and M_3 are defined as

$$M_1 := U_{x_1}^{-1} V_{x_1}^{-1} W_{x_1}^{-1}$$

$$M_2 := U_{x_2}^{-1} V_{x_2}^{-1} W_{x_2}^{-1}$$

$$M_3 := U_{x_3}^{-1} V_{x_3}^{-1} W_{x_3}^{-1}$$

The superscript T denoting the 'transpose' is illustrated in Figure 3.

We can readily generalize (61) and (62) to the higher multivariate cases. With $U_{x_i}^{-1}$, V_{xi}^{-1} and W_{xi}^{-1} having the usual meanings, define M_i for a unit box domain as

$$M_i := U_{x_i}^{-1}$$

and for a general box domain as

$$M_i := U_{x_i}^{-1} V_{x_i}^{-1} W_{x_i}^{-1}$$

Generalizing (61) and (62) to l variables, we easily get the formula for computing the Bernstein coefficients as

$$B = \left(M_l \ldots \left(M_i \ldots \left(M_3 \left(M_2 \left(M_1 A \right)^T \right)^T \right)^{T \ldots} \right)^{T \ldots} \right)^T \tag{63}$$

where A is the polynomial coefficients matrix with $(n+1)$ rows and $(n+1)^{l-1}$ columns. The superscript T denotes the 'transpose', which means here changing the order of the indices of the elements of the l dimensional array cyclically, in the reverse order and finally converting the elements of first coordinate direction to the l^{th} coordinate direction.

Since M_i and A are matrices, their products would also be matrices. In this form of representation, every 'transpose' denoted by the superscript T in (63) amounts to transposing and proper reshaping of the resulting matrix. These operations can very easily be carried out using 'transpose' and 'reshape' commands of popular programming languages such as FORTRAN 95. Theoretically, polynomials of any dimension and degree can be readily handled using this *matrix* representation, since the polynomial coefficients array A can be always expressed in the form of a matrix, and only matrix operations such as multiplication, inverse, transpose and reshape operations are involved.

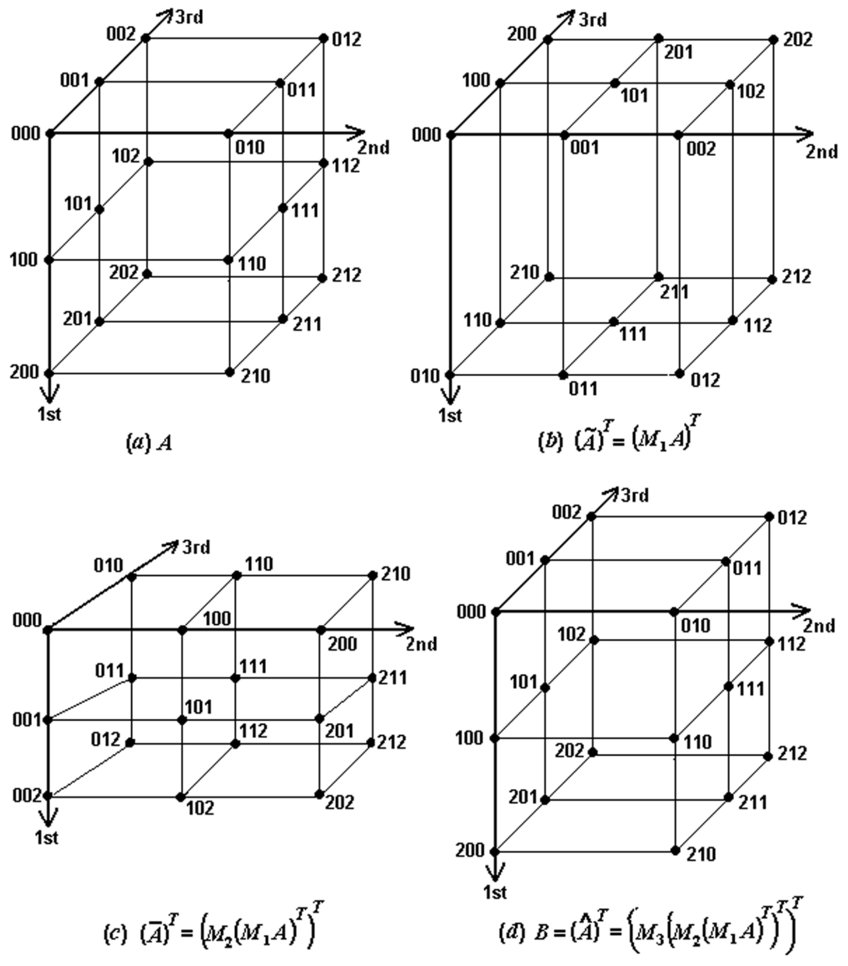
(a) A
(b) $(\tilde{A})^T = (M_1 A)^T$
(c) $(\bar{A})^T = \left(M_2 (M_1 A)^T\right)^T$
(d) $B = (\hat{A})^T = \left(M_3 \left(M_2 (M_1 A)^T\right)^T\right)^T$
1st
2nd
3rd

Figure 3: Rotation of axes for a three-dimensional array example with $n_1 = 2$, $n_2 = 1$ and $n_3 = 2$: (a) Coefficient matrix A , (b) First transpose $(\widetilde{A})^T = (M_1A)^T$, (c) Second transpose $(\overline{A})^T = \left(M_2\,(M_1A)^T\right)^T$, (d) Third transpose giving the Bernstein coefficient matrix $B = \left(M_3\left(M_2\,(M_1A)^T\right)^T\right)^T$

5 Proposed Algorithm

Based on the above developments, we give two algorithms using the proposed *Matrix* method for computing the Bernstein coefficients. The first Algorithm Bernstein_Matrix_unit computes the Bernstein coefficients specifically on a unit box domain, whereas the second Algorithm Bernstein_Matrix computes the Bernstein coefficients on a general box domain. Since on a unit box domain the product matrix $M := \widetilde{U}_x$ becomes a lower triangular matrix, the former algorithm is a simplified algorithm for Bernstein coefficient computation on a unit domain requiring fewer arithmetic operations.

We assume in the algorithms that the degree N of all the variables are equal to n, purely for convenience of presentation. The elements of the input coefficient matrix A are first arranged as $a_{i,k}$, $i = 0, ..., n, k = 0, ..., ((n+1)^{l-1} - 1)$, and supplied as an input to the algorithm.

Algorithm Bernstein_Matrix_unit : $B(\mathbf{u})$ = Bernstein_Matrix_unit $(N,\, l,\, A)$

Inputs : Degree N of the polynomial p (N is taken to be n here), number of variables l, and coefficient matrix A whose elements are the polynomial coefficients a_I.

Output : A patch $B(\mathbf{u})$ of Bernstein coefficients of p. Note that B is output in the form of a matrix.

```
BEGIN Algorithm
1. {Compute the binomial coefficients}
   $C(0:n, 0:n)$ = Binomial_coefficient $(n)$.
2. {Compute inverse of $U_x$}
   $\widetilde{U}_x(0:n, 0:n)$ = InverseUx $(n,\, C(0:n, 0:n))$
   $M = \widetilde{U}_x$
3. {Iterate}
   for $r = 1$ to $l$ do
      1. $A = MA$
      2. transpose $A$
      3. reshape $A$ to the required matrix shape.
4. {Return}
   return $B(\mathbf{u}) = A$.
END Algorithm
```

Computational complexity of Algorithm Bernstein_Matrix_unit

- Step 1 requires $n(n-1)$ multiplications and additions (refer equation 16).
- Step 2 requires $n(n-1)/2$ multiplications (refer equation 45) and no additions.
- Since on a unit box domain M is a lower triangular matrix with all the elements of its first column as unity, step 3.1 requires $ln(n+1)^l/2$ additions and multiplications.

- Summing up from steps 2 to 3 (i.e., without considering the operations involved in binomial coefficient computations), the total additions is

$$l\frac{n(n+1)^l}{2} \tag{64}$$

and the total multiplications is

$$\frac{n(n-1)}{2} + l\frac{n(n+1)^l}{2} \tag{65}$$

Algorithm Bernstein_Matrix : $B(\mathbf{u})$ = Bernstein_Matrix (N, l, $\mathbf{x}$, A)

Inputs : Degree N of the polynomial p (N is taken to be n here), number of variables l, the l -dimensional domain box $\mathbf{x}$, and coefficient matrix A whose elements are the polynomial coefficients a_I.

Output : A patch $B(\mathbf{u})$ of Bernstein coefficients of p. Note that B is output in the form of a matrix.

```
BEGIN Algorithm
1. {Compute the binomial coefficients}
   $C(0:n, 0:n)$ = Binomial_coefficient ($n$).
2. {Compute inverse of $U_x$}
   $\widetilde{U}_x(0:n, 0:n)$ = InverseUx ($n$, $C(0:n, 0:n)$)
3. {Iterate}
   for $r = 1$ to $l$ do
      1. {Compute inverse of $V_x$}
         $\widetilde{V}_x(0:n, 0:n)$ = InverseVx ($n$, $\mathbf{x}$).
      2. {Compute inverse of $W_x$}
         $\widetilde{W}_x(0:n, 0:n)$ = InverseWx ($n$, $\mathbf{x}$, $C(0:n, 0:n)$).
      3. {Product of all the inverse matrices}
         $M_r$ =ProductM ($\widetilde{U}_x, \widetilde{V}_x, \widetilde{W}_x$).
4. {Iterate}
   for $r = 1$ to $l$ do
      1. $A = M_r A$
      2. transpose $A$
      3. reshape $A$ to the required matrix shape.
5. {Return}
   return $B(\mathbf{u}) = A$.
END Algorithm
```

Computational complexity of Algorithm Bernstein_Matrix

- Step 1 requires $n(n-1)$ multiplications and additions (refer equation 16).
- Step 2 requires $n(n-1)/2$ multiplications (refer equation 45) and no additions.

- Step 3.1 is executed l times, hence it requires l additions and ln multiplications.
- Step 3.2 requires $ln(n+3)/2$ multiplications and no additions (refer equation 46).
- Step 3.3 requires
$$l \sum_{i=1}^{n} i\,(2(n-i)+1)$$
additions (refer equation 47) and
$$l\left(\frac{n(n+1)}{2} + \sum_{i=1}^{n+1} i\,(2(n+1-i)+1)\right)$$
multiplications (refer equation 48).
- Step 4.1 requires $ln(n+1)^l$ additions and $l(n+1)^{l+1}$ multiplications.
- Summing up from steps 2 to 4 (i.e., neglecting the work involved in binomial coefficient computations), the total additions is
$$l + l\sum_{i=1}^{n} i\,(2(n-i)+1) + ln(n+1)^l \tag{66}$$
while the total multiplications is
$$\frac{n(n-1)}{2} + ln + \frac{ln(n+3)}{2} + l\frac{n(n+1)}{2} + l\sum_{i=1}^{n+1} i\,(2(n+1-i)+1) + l(n+1)^{l+1} \tag{67}$$

6 Computational Complexity and Memory Requirements

In this section we compare the computational complexities of the Conventional method and Garloff's method derived in Sections 3.1 and 3.3 respectively, with that of the proposed method. In Table 2, we give the arithmetic involved in all the three methods on a unit box domain, assuming that all the binomial coefficients have been precomputed and stored.

Table 2: Unit box domain : Number of arithmetic operations required to compute the Bernstein coefficients with the three methods

Method	Number of additions	Number of multiplications	Total
Conventional	$\left(\frac{(n+1)(n+2)}{2}\right)^l$	$2l\left(\frac{(n+1)(n+2)}{2}\right)^l$	$O(n^{2l})$
Garloff's	$l\frac{n(n+1)^l}{2}$	$l(n+1)^l$	$O(n^{l+1})$
Matrix	$l\frac{n(n+1)^l}{2}$	$\frac{n(n-1)}{2} + l\frac{n(n+1)^l}{2}$	$O(n^{l+1})$

From Table 2, we see that on a unit box domain, the computational complexity of the Conventional method is $O(n^{2l})$, whereas those of Garloff's method and the proposed method is $O(n^{l+1})$.

Next, assuming that all the binomial coefficients are precomputed, the arithmetic involved in all the three methods for evaluating Bernstein coefficients on a general box domain are reported in Table 3. The table also reports the computations involved in transforming the polynomial from a general box domain to a unit box domain using the multivariate Horner scheme. For one term of the polynomial, the computational complexity of this transformation onto a unit box is $O(n^l)$ (see Remark 1); and for all the terms of the polynomial, it is $O(n^{2l})$ (see Remark 2).

Table 3: General box domain : Number of arithmetic operations required by Horner scheme to transform all the polynomial coefficients, and by the three methods for computing the Bernstein coefficients

Method	Number of additions	Number of multiplications	Total
Horner scheme	$l + \left(\frac{(n+1)(n+2)}{2}\right)^l$ $- (n+1)^l$	$ln + 2\left(\frac{(n+1)(n+2)}{2}\right)^l$ $-2(n+1)^l + l(n+1)^l$	$O(n^{2l})$
Conven-tional	$l + \left(\frac{(n+1)(n+2)}{2}\right)^l$ $- (n+1)^l$ $+ \left(\frac{(n+1)(n+2)}{2}\right)^l$	$ln + 2\left(\frac{(n+1)(n+2)}{2}\right)^l$ $-2(n+1)^l + l(n+1)^l$ $+2l\left(\frac{(n+1)(n+2)}{2}\right)^l$	$O(n^{2l})$
Garloff's	$l + \left(\frac{(n+1)(n+2)}{2}\right)^l$ $- (n+1)^l + l\frac{n(n+1)^l}{2}$	$ln + 2\left(\frac{(n+1)(n+2)}{2}\right)^l$ $-2(n+1)^l + 2l(n+1)^l$	$O(n^{2l})$
Matrix	$l + l\sum\limits_{i=1}^{n} i\,(2(n-i)+1)$ $+ln(n+1)^l$	$\frac{n(n-1)}{2} + ln + \frac{ln(n+3)}{2}$ $+l\frac{n(n+1)}{2} + l(n+1)^{l+1}$ $+ l\sum\limits_{i=1}^{n+1} i\,(2(n+1-i)+1)$	$O(n^{l+1})$

From Table 3, we note that on a general box domain, the computational complexity of the Conventional method (see remark 3) and Garloff's method (see remark 5) is $O(n^{2l})$, whereas that of the proposed method is $O(n^{l+1})$.

For some selected values of degree n and number of variables l, we calculate the number of additions, the number of multiplications and the total number of operations required by Garloff's method and the proposed method on both the unit and general box domains. These are reported respectively in Table 4 and Table 5[1]. For a unit box domain, the ratio of the total number of operations required by Garloff's method to that of the proposed method is found to be

$$\frac{l(n+1)^l(n/2+1)}{n(n-1)/2 + ln(n+1)^l}$$

For asymptotically large values of n or l, this ratio converges to 0.5. We also find the values of this ratio for the respective degree and dimension, and report the same

[1] In these tables, we show a comparison only between these two methods, as the Conventional method requires much larger number of arithmetic operations than Garloff's method, and so is ignored.

in the last column of Table 4. In case of a general box domain, for the fixed degree given in column 1, we report in the last two columns of Table 5, the minimum and the maximum values of the ratios, i.e. for dimensions 4 and 6, respectively.

Table 4: Unit box domain : Comparison of the number of arithmetic operations required to compute Bernstein coefficients with Garloff's Method and the proposed Matrix method

Degree n	Operations	Garloff's method			Matrix method			Ratio
		Dimension l			Dimension l			
		4	5	6	4	5	6	
2	additions	324	1215	4374	324	1215	4374	
	multiplications	324	1215	4374	325	1216	4375	
	total	648	2430	8748	649	2431	8749	1.0
3	additions	1536	7680	36864	1536	7680	36864	
	multiplications	1024	5120	24576	1539	7683	36867	
	total	2560	12800	61440	3075	15363	73731	0.83
6	additions	28812	2.5e05	2.1e06	28812	2.5e05	2.1e05	
	multiplications	9604	8.4e04	7.0e05	28827	2.5e05	2.1e05	
	total	38416	3.4e05	2.8e06	57639	5.0e05	4.2e06	0.67
7	additions	5.7e04	5.7e05	5.5e06	5.7e04	5.7e05	5.5e06	
	multiplications	1.6e04	1.6e05	1.6e06	5.7e04	5.7e05	5.5e06	
	total	7.4e04	7.4e05	7.1e06	1.1e05	1.1e06	1.1e07	0.64

From Table 4, we see that the total number of operations required to compute the Bernstein coefficients on a unit box domain is less with Garloff's method than with the proposed Matrix method. Keeping the degree n fixed and varying the dimension l, the ratio of the total number of operations required by Garloff's method to that of the proposed method remains the same. In the limiting case where n or l tends to infinity, the ratio becomes 0.5.

In contrast, from Table 5, for a general box domain, we see that the total number of operations required to compute the Bernstein coefficients is much less with the proposed Matrix method than with Garloff's method. For a given degree, the ratio of the total number of operations required by Garloff's method to that of the proposed method increases with the dimension. Further, the proposed method also becomes relatively much more efficient with the degree, as can be seen from the entries of the last two columns of the table. Hence, we may conclude that the proposed Matrix method becomes increasingly more computationally efficient than the existing methods, when the number of Bernstein coefficients to be computed increases.

As far as memory requirements or array storage is concerned, the Conventional method, Garloff's method and the proposed Matrix method require one array of size $(n+1)^l$ for computing and storing the Bernstein coefficients.

7 Numerical Experiments

Using the three methods of Bernstein coefficient computation discussed above, we compute the Bernstein coefficients for eleven real world problems, and compare the

Table 5: General box domain : Comparison of the number of arithmetic operations required to compute Bernstein coefficients with Garloff's Method and the proposed Matrix method

Deg n	Operations	Garloff's method			Matrix method			Min ratio	Max ratio
		Dimension l			Dimension l				
		4	5	6	4	5	6		
2	adds	1.5403	8.7503	5.0e4	6.72e2	2.46e3	8.78e3		
	mults	3.0803	1.75e4	1.00e5	1.07e3	3.77e3	1.32e4		
	total	4.62e3	2.62e4	1.50e5	1.74e3	6.23e3	2.20e4	2.65	6.84
3	adds	1.12e4	1.06e5	1.03e6	3.13e3	1.54e4	7.38e4		
	mults	2.15e4	2.08e5	2.04e6	4.30e3	2.07e4	9.86e4		
	total	3.28e4	3.14e5	3.07e6	7.43e3	3.61e4	1.72e5	4.42	17.82
6	adds	6.41e0	1.74e7	4.84e8	5.79e4	5.04e5	4.23e6		
	mults	1.24e6	3.45e7	9.66e8	6.80e4	5.89e5	4.9e6		
	total	1.88e6	5.20e7	1.45e9	1.26e5	1.09e6	9.17e6	14.95	158
7	adds	1.73e6	6.10e7	2.18e9	1.15e5	1.14e6	1.10e7		
	mults	3.38e6	1.21e8	4.36e9	1.32e5	1.31e6	1.25e7		
	total	5.11e6	1.82e8	6.54e9	2.47e5	2.45e6	2.35e7	20.68	278

computational time taken. The eleven problems are taken from [14] and described in Appendix A. A time limit of five hours is imposed for the computations, while the available computer memory is limited to 2 GB RAM. All computations are performed on a Sun 440 MHz Ultra Sparc 10 Workstation with the codes developed in Forte FORTRAN 95 [?]. All rounding errors are accounted for by using interval arithmetic support provided in the compiler.

For a *unit* box domain, Table 6 gives the computation time taken (in seconds) to compute the Bernstein coefficients by the three methods for the eleven problems. In the table, we have also reported the degree (as 'deg') of each problem considered in column 4, and the number of Bernstein coefficients required to be computed in column 5.

For a *general* box domain given in Appendix A, Table 7 gives the computation time taken (in seconds) to compute the Bernstein coefficients by the three methods for the eleven problems. In column 5 of this table, we also report the time taken (in seconds) only for transforming the polynomial coefficients from a general box domain to a unit box domain, using the multivariate Horner's scheme described in Section 3.1. Recall that the transformed polynomial coefficients are needed to compute the Bernstein coefficients in Garloff's method and the Conventional method. The times reported in columns 6 and 7 already include this time taken by Horner's scheme.

For the purpose of comparison, we also report the values of the *percent reduction* in Tables 6 and 7. This is computed as

$$\frac{\text{Time taken by existing method} - \text{Time taken by proposed matrix method}}{\text{Time taken by existing method}} \times 100$$

where, the 'existing method' is either the Conventional method or Garloff's method, as the case maybe.

Table 6: Unit box domain : Computation times taken by Garloff's method, Conventional method and Matrix method to compute Bernstein coefficients

Ex	Test function	Dim	deg	No. of Bern coeff	Time taken in seconds by			% reduction	
					Garloff	Conv	Matrix	Garloff	Conv
1.	L. V. 3	3	2	18	0.0002	0.0004	0.0006	-216.634	-76.021
2.	R. D. 3	3	2	12	0.0002	0.0002	0.0006	-252.878	-264.939
3.	L. V. 4	4	2	54	0.0004	0.0030	0.0007	-108.554	74.973
4.	Cap 4	4	3	64	0.0004	0.0042	0.0009	-104.809	79.367
5.	Wrig 5	5	2	48	0.0003	0.0027	0.0007	-145.389	72.460
6.	Reim 5	5	6	16,807	0.0556	140.01	0.0338	39.147	99.976
7.	Mag 6	6	2	729	0.0020	0.3307	0.0021	-5.904	99.359
8.	But 6	6	3	288	0.0010	0.0667	0.0012	-19.980	98.160
9.	Reim 6	6	7	262,144	1.3545	17075.2	0.7175	46.393	99.996
10.	Mag 7	7	2	2,187	0.0064	2.2973	0.0059	7.973	99.743
11.	Reim 7	7	8	4,782,969	52.381	*	21.8375	58.310	*

'*' entry denotes that computation was incomplete at the end of time limit of five hours.

Table 7: General box domain : Computation times taken for transformation to unit domain by Horner's method, and for computation of Bernstein coefficients by various methods

Ex	Test Function	Dim	deg	Time Horner	Time taken in seconds by			%cent reductions	
					Garloff	Conv	Matrix	Garloff	Conv
1.	L.V.3	3	2	0.0007	0.0009	0.0011	0.0008	16.706	29.323
2.	R.D.3	3	2	0.0007	0.0009	0.0009	0.0008	14.952	14.376
3.	L. V. 4	4	2	0.0009	0.0013	0.0039	0.0009	31.796	77.241
4.	Cap 4	4	3	0.0009	0.0013	0.0051	0.0009	30.107	81.846
5.	Wrig 5	5	2	0.0008	0.0011	0.0035	0.0009	19.822	74.097
6.	Reim 5	5	6	0.0410	0.0971	140.05	0.0341	64.908	99.976
7.	Mag 6	6	2	0.0026	0.0046	0.3332	0.0022	51.122	99.327
8.	But 6	6	3	0.0016	0.0026	0.0683	0.0014	47.456	97.968
9.	Reim 6	6	7	0.8430	2.1976	17076	0.7175	67.351	99.996
10.	Mag 7	7	2	0.0072	0.0136	2.3045	0.0063	53.815	99.727
11.	Reim 7	7	8	24.822	77.203	*	22.2797	71.141	*

'*' entry denotes that computation was incomplete due to time/memory restrictions

From the tables, we note at the outset that in the specified time limit of five hours, with Garloff's method and the Matrix method we are able to compute the Bernstein coefficients for all the problems, whereas with the Conventional method we are successful in all but one problem. On a unit box domain, the reduction in the computational time with the Matrix method over the Conventional method varies from -264.94% to 99.99%, while on a general box domain the reduction varies from 14.38% to 99.99%. The Conventional method is seen to be relatively quite slow. Hence, we shall next focus only on comparing the Matrix method with Garloff's method.

To aid in the comparative analysis, we extract and summarize the results as a function of the number of Bernstein coefficients in Table 8. From Table 8, we see that on a unit box domain the reduction in the computational time with the Matrix method over Garloff's method varies from -252.88% to 58.31%, while on a general box domain the reduction varies from 14.95% to 71.14%. We also observe that the Matrix method becomes relatively faster with the number of Bernstein coefficients computed. The relative speed improvements with the Matrix method can be attributed to a fuller exploitation of efficient matrix operations, as compared to Garloff's method. This exploitation naturally yields relatively better computational time as the number of coefficients 'increases'.

Table 8: Percent reduction in computational time obtained with the proposed Matrix method over Garloff's method, as function of the number of Bernstein coefficients computed

S. No	Number of Bernstein coefficients	Unit box domain	General box domain
1.	12	-252.8785	14.9523
2.	18	-216.6344	16.7063
3.	48	-145.3887	19.8220
4.	54	-108.5537	31.7959
5.	64	-104.8091	30.1069
6.	288	-19.9804	47.4597
7.	729	-5.9041	51.1220
8.	2,187	7.9726	53.8151
9.	16,807	39.1474	64.9080
10.	262,144	46.3931	67.3507
11.	4,782,969	58.3100	71.1414

8 Conclusions

In this paper, we proposed a *Matrix* method to compute the Bernstein coefficients of a multivariate polynomial on a general box-like domain. The proposed method involves only matrix operations. This enables one to take advantage of the built-in array operation routines of programming languages to accelerate the computations. Moreover, on a general box-like domain, the proposed method has a lesser computational complexity than the existing methods. Numerical tests conducted on eleven polynomial test problems over unit and general box domain show the proposed method to yield significant reductions in computational time over existing methods as the number of Bernstein coefficients becomes 'larger'.

Acknowledgments

The authors thank Prof. Jürgen Garloff of University of Applied Sciences in Constance, Germany for his comments and suggestions on the work reported in this paper.

A Description of Test Problems

In the following, we list the polynomials p, the domain boxes $\mathbf{x}$, the abbreviated and full names, and the dimensionality of the problems considered in our tests. The problems are arranged in the order of increasing dimensionality. All these test problems are taken from Verschelde's PHC pack [14].

1. **L. V. 3** : A neural network modeled by an adaptive Lotka-Volterra system, $l = 3$

$$p(x_1, x_2, x_3) = x_1 {x_2}^2 + x_1 x_3^2 - 1.1x_1 + 1$$

$$\mathbf{x_1} = [-1.5, 2],\ \mathbf{x_2} = [-1.5, 2],\ \mathbf{x_3} = [-1.5, 2]$$

2. **R. D. 3** : A 3-dimensional reaction diffusion problem, $l = 3$

$$p(x_1, x_2, x_3) = x_1 - 2x_2 + x_3 + .835634534x_2(1 - x_2)$$

$$\mathbf{x_1} = [-5, 5],\ \mathbf{x_2} = [-5, 5],\ \mathbf{x_3} = [-5, 5]$$

3. **L. V.** 4 : A neural network modeled by an adaptive Lotka-Volterra system, $l = 4$

$$p(x_1, x_2, x_3, x_4) = x_1 x_2^2 + x_1 x_3^2 + x_1 x_4^2 - 1.1x_1 + 1$$

$$\mathbf{x_1} = [-2, 2],\ \mathbf{x_2} = [-2, 2],\ \mathbf{x_3} = [-2, 2],\ \mathbf{x_4} = [-2, 2]$$

4. **Cap 4 :** Caprasse's system **:** $l = 4$

$$\begin{aligned} p(x_1, x_2, x_3, x_4) &= -x_1 x_3^3 + 4x_2 x_3^2 x_4 + 4x_1 x_3 x_4^2 + 2x_2 x_4^3 \\ &\quad +4x_1 x_3 + 4x_3^2 - 10x_2 x_4 - 10x_4^2 + 2 \end{aligned}$$

$$\mathbf{x_1} = [-.5, .5],\ \mathbf{x_2} = [-.5, .5],\ \mathbf{x_3} = [-.5, .5],\ \mathbf{x_4} = [-.5, .5]$$

5. **Wrig** 5 : System of A.H. Wright, $l = 5$

$$p(x_1, x_2, x_3, x_4, x_5) = x_5^2 + x_1 + x_2 + x_3 + x_4 - x_5 - 10$$

$$\begin{aligned} \mathbf{x_1} &= [-5, 5],\ \mathbf{x_2} = [-5, 5],\ \mathbf{x_3} = [-5, 5],\ \mathbf{x_4} = [-5, 5], \\ \mathbf{x_5} &= [-5, 5] \end{aligned}$$

6. **Reim 5**: The 5-dimensional system of Reimer, $l = 5$

$$p(x_1, x_2, x_3, x_4, x_5) = -1 + 2x_1^6 - 2x_2^6 + 2x_3^6 - 2x_4^6 + 2x_5^6$$

$$\mathbf{x_1} = [-1, 1],\ \mathbf{x_2} = [-1, 1],\ \mathbf{x_3} = [-1, 1],\ \mathbf{x_4} = [-1, 1],\ \mathbf{x_5} = [-1, 1]$$

7. **Mag 6** : A problem of magnetism in physics, $l = 6$

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = 2x_1^2 + 2x_2^2 + 2x_3^2 + 2x_4^2 + 2x_5^2 + x_6^2 - x_6$$

$$\begin{aligned} \mathbf{x_1} &= [-5, 5],\ \mathbf{x_2} = [-5, 5],\ \mathbf{x_3} = [-5, 5],\ \mathbf{x_4} = [-5, 5], \\ \mathbf{x_5} &= [-5, 5],\ \mathbf{x_6} = [-5, 5] \end{aligned}$$

8. **But 6** : Butcher's problem, $l = 6$

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = x_6 x_2^2 + x_5 x_3^2 - x_1 x_4^2 + x_4^3 + x_4^2 - 1/3 x_1 + 4/3 x_4$$

$$\begin{aligned} \mathbf{x_1} &= [-1, 0],\ \mathbf{x_2} = [-.1, .9],\ \mathbf{x_3} = [-.1, .5], \mathbf{x_4} = [-1, -.1], \\ \mathbf{x_5} &= [-.1, -.05],\ \mathbf{x_6} = [-.1, -.03] \end{aligned}$$

9. **Reim 6** : The 6-dimensional system of Reimer, $l = 6$

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = -1 + 2x_1^7 - 2x_2^7 + 2x_3^7 - 2x_4^7 + 2x_5^7 - 2x_6^7$$

$$\begin{aligned} \mathbf{x_1} &= [-5, 5],\ \mathbf{x_2} = [-5, 5],\ \mathbf{x_3} = [-5, 5],\ \mathbf{x_4} = [-5, 5], \\ \mathbf{x_5} &= [-5, 5],\ \mathbf{x_6} = [-5, 5] \end{aligned}$$

10. **Mag 7** : Katsura 6, a problem of magnetism in physics, $l = 7$

$$p(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = x_1^2 + 2x_2^2 + 2x_3^2 + 2x_4^2 + 2x_5^2 + 2x_6^2 + 2x_7^2 - x_1$$

$$\begin{aligned} \mathbf{x_1} &= [-5, 5],\ \mathbf{x_2} = [-5, 5],\ \mathbf{x_3} = [-5, 5],\ \mathbf{x_4} = [-5, 5], \\ \mathbf{x_5} &= [-5, 5],\ \mathbf{x_6} = [-5, 5],\ \mathbf{x_7} = [-5, 5] \end{aligned}$$

11. **Reim 7** : The 7-dimensional system of Reimer, $l = 7$

$$p(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = -1 + 2x_1^8 - 2x_2^8 + 2x_3^8 - 2x_4^8 + 2x_5^8 - 2x_6^8 + 2x_7^8$$

$$\begin{aligned} \mathbf{x_1} &= [-1, 1],\ \mathbf{x_2} = [-1, 1],\ \mathbf{x_3} = [-1, 1],\ \mathbf{x_4} = [-1, 1], \\ \mathbf{x_5} &= [-1, 1],\ \mathbf{x_6} = [-1, 1],\ \mathbf{x_7} = [-1, 1] \end{aligned}$$

References

[1] J. Berchtold, I. Voiculescu, and A. Bowyer. Multivariate Bernstein form polynomials. Technical Report 31/98, School of Mechanical Engineering, University of Bath, 1998.

[2] Z. A. Garczarczyk. Parallel schemes of computation for bernstein coefficients and their application. In *Proc. of the International Conference on Parallel Computing in Electrical Engineering*, pages 334–337, Warsaw, Poland, 2002. IEEE Computer Society.

[3] J. Garloff. Convergent bounds for the range of multivariate polynomials. In K. Nickel, editor, *Interval Mathematics*, volume 212 of *Lecture Notes in Computer Science*, pages 37–56. Springer, Berlin, 1985.

[4] J. Garloff. The Bernstein algorithm. *Interval Computations*, 2:154–168, 1993.

[5] J. Garloff and A. P. Smith. Solution of systems of polynomial equations by using Bernstein expansion. In G. Alefeld, J. Rohn, S. Rump, and T. Yamamoto, editors, *Symbolic Algebraic Methods and Verification Methods*, pages 87–97. Springer, New York, 2001.

[6] J. M. Lane and R. F. Riesenfeld. Bounds on a polynomial. *BIT*, 21(1):112–117, 1981.

[7] J. M. Pena and T. Sauer. On the multivariate Horner scheme. *SIAM Journal of Numerical Analysis*, 37(4):1186–1197, 2000.

[8] H. Ratschek and J. Rokne. *Computer methods for the range of functions*. Ellis Horwood, New York (Chichester), 1984.

[9] J. Rokne. Bounds for an interval polynomial. *Computing*, 18:225–240, 1977.

[10] J. Sànchez-Reyes. Algebraic manipulation in the Bernstein form made simple via convolutions. *Computer-Aided Design*, 35(10):959–967, September 2003.

[11] A. P. Smith. Fast construction of constant bound functions for sparse polynomials. *Journal of Global Optimization*, 43(2–3):445–458, 2009.

[12] V. Stahl. *Interval methods for bounding the range of polynomials and solving systems of nonlinear equations*. PhD thesis, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, Austria, 1995.

[13] B. Sunar and D. Cyganski. Comparison of bit and word level algorithms for evaluating unstructured functions over finite rings. In K. Nickel, editor, *Cryptographic Hardware and Embedded Systems CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 237–249. Springer, Berlin, 2005.

[14] J. Verschelde. The PHC pack, the database of polynomial systems. Technical report, University of Illinois, Mathematics Department, Chicago, U.S.A., 2001.