Annales UMCS Informatica AI XII, 2 (2012) 7–18
DOI: 10.2478/v10065-012-0002-6

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Fast multidimensional Bernstein-Lagrange algorithms

Joanna Kapusta[1*], Ryszard Smarzewski[1†]

[1] *Institute of Mathematics and Computer Science,*
The John Paul II Catholic University of Lublin,
ul. Konstantynow 1H, 20-708 Lublin, Poland

Abstract – In this paper we present two fast algorithms for the Bézier curves and surfaces of an arbitrary dimension. The first algorithm evaluates the Bernstein-Bézier curves and surfaces at a set of specific points by using the fast Bernstein-Lagrange transformation. The second algorithm is an inversion of the first one. Both algorithms reduce the initial problem to computation of some discrete Fourier transformations in the case of geometrical subdivisions of the d-dimensional cube. Their orders of computational complexity are proportional to those of corresponding d-dimensional FFT-algorithm, i.e. to $O(N \log N) + O(dN)$, where N denotes the order of the Bernstein-Bézier curves.

1 Introduction

Let $n = (n_1, n_2, \ldots, n_d)$ be a d-tuple of positive integers and K be a field. Moreover, let Q_n be a lattice of all $N = n_1 n_2 \ldots n_d$ multi-indices $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d)$ with the integer coordinates satisfying inequalities

$$0 \le \alpha_i < n_i \text{ for } i = 1, 2, \ldots, d. \tag{1}$$

Using the multi-index notation, we write Bernstein-Bézier vector polynomials of the variable $x = (x_1, x_2, \ldots, x_d) \in K^d$ in the form

$$p_n(x) = \sum_{\alpha \in Q_n} f_\alpha B_\alpha(x), \tag{2}$$

*jkapusta@kul.lublin.pl
†rsmax@kul.lublin.pl

where $f_\alpha \in K^s$ are the control points, the summation extends over all $n_1 n_2 \cdots n_d$ multi-indices α from the lattice Q_n, and

$$B_\alpha(x) = \binom{n-1}{\alpha} x^\alpha (1-x)^{n-\alpha-1} = \prod_{j=1}^{d} \binom{n_j - 1}{\alpha_j} x_j^{\alpha_j} (1-x_j)^{n_j-1-\alpha_j}, \tag{3}$$

where $n-1 = (n_1 - 1, n_2 - 1, \dots, n_d - 1)$. Note that $p_n(x)$ is a Bézier curve or surface in the case when $d = 1$ or $d = 2$, respectively.

Additionally, suppose that

$$x_\alpha = (x_{1,\alpha_1}, x_{2,\alpha_2}, \dots, x_{d,\alpha_d}), \ \alpha = (\alpha_1, \alpha_2, \dots, \alpha_d) \in Q_n, \tag{4}$$

are the points in K^d such that coordinates

$$x_{i,0}, x_{i,1}, \dots, x_{i,n_i-1} \quad (i = 1, 2, \dots, d) \tag{5}$$

are pairwise distinct, i.e. $x_{i,j} \neq x_{i,k}$, whenever $j \neq k$. Then the Bernstein-Bézier vector polynomial $p_n(x)$ can be written in the Lagrange form

$$p_n(x) = \sum_{\alpha \in Q_n} y_\alpha L_\alpha(x), \tag{6}$$

where $y_\alpha = p_n(x_\alpha) \in K^s$ and

$$L_\alpha(x) = \prod_{i=1}^{d} L_{\alpha_i}(x_i), \ L_{\alpha_i}(x_i) = \prod_{\substack{j=0 \\ j \neq \alpha_i}}^{n_i-1} \frac{x_i - x_{i,j}}{x_{i,\alpha_i} - x_{i,j}}. \tag{7}$$

In this paper we present two fast algorithms of the order

$$O(N \log N) + O(dN), \ N = |Q_n| = n_1 n_2 \dots n_d, \tag{8}$$

for the Bernstein-Lagrange transformation $\mathcal{T} : (f_\beta)_{\beta \in Q_n} \to (y_\beta)_{\beta \in Q_n}$, and its inverse, which is defined by

$$\mathcal{T} : y_\beta = \sum_{\alpha \in Q_n} f_\alpha B_\alpha(x_\beta), \ \beta \in Q_n, \tag{9}$$

where the coordinates of points x_β are such that

$$x_{i,j} = \lambda_i \gamma_i^j \quad (i = 1, 2, \dots, d, \ j = 0, 1, \dots, n_i - 1) \tag{10}$$

with the scalars $\lambda_i \neq 0$, $\gamma_i \neq 0$ and $\gamma_i \neq 1$ $(i = 1, 2, \dots, d)$ fixed in K. For the simplicity, these algorithms will be established under the additional assumption that $s = 1$, which does not restrict the generality of our considerations.

Since the coordinates $x_{i,j}$ $(j = 0, 1, \dots, n_i - 1)$ form geometrical progression, it follows that the points x_β can be used e.g. in extrapolation problems [**1**]. It is not clear if it is possible to extend our fast algorithms to the case of arithmetic progression, or more generally to the case when

$$x_{i,j} = \lambda_i x_{i,j-1} + \delta_i \ (i = 1, 2, \dots, d, \ j = 1, 2, \dots, n_i - 1), \tag{11}$$

where $\lambda_i \neq 0$, δ_i and $x_{i,0} = \varkappa_i$ belong to the field K [**2**]. Of course, in order to evaluate the transformation $\mathcal{T}$ for the last coordinates one can use multidimensional algorithms based on the de Casteljau algorithm, which have the computational complexity of the order greater than our algorithms, cf. [**3**], [**4**], [**5**] and [**6**].

Following [**7**] and [**8**], our algorithms will use the discrete Fourier transformation

$$F_m\colon K^m \ni a \to b \in K^m \tag{12}$$

and its inverse, which are defined by

$$b_i = \sum_{k=0}^{m-1} a_k \psi_m^{ik} \quad \text{and} \quad a_i = \frac{1}{m}\sum_{k=0}^{m-1} b_k \psi_m^{-ik} \quad (i = 0, 1, \ldots, m-1),$$

where ψ_m is supposed to be a primitive root of the unity of order m in the field K. It is well known that discrete Fourier transformations can be computed by the famous FFT-algorithm, which has a running time of order $O(m \log m)$ [**9**].

2 Fast multidimensional convolutions and deconvolutions

In order to present fast algorithms for computation of the Bernstein-Lagrange transformations $\mathcal{T}$ and $\mathcal{T}^{-1}$, we need fast algorithms for multidimensional convolutions and deconvolutions. For this purpose, suppose that $a = (a_0, a_1, \ldots)$ and $b = (b_0, b_1, \ldots)$ are two finite or infinite sequences. Then the wrapped convolution

$$c = (c_0, c_1, \ldots, c_{m-1}) = a \otimes_m b \tag{13}$$

is defined by

$$c_i = \sum_{k=0}^{i} a_k b_{i-k} \quad (i = 0, 1, \ldots, m-1)\,, \tag{14}$$

while its deconvolution

$$a = c \oslash_m b = c \otimes_m b^{-1} \;\; (b_0 \neq 0)$$

is supposed to be the solution

$$\begin{aligned} a_0 &= c_0/b_0, \\ a_i &= \left(c_i - \sum_{k=0}^{i-1} a_k b_{i-k}\right)/b_0 \;\; (i = 1, 2, \ldots, m-1) \end{aligned} \tag{15}$$

of the lower triangular system of equations (14). Moreover, the convolutionary inverse

$$d = (d_0, d_1, \ldots d_{r-1}) = b^{-1} \tag{16}$$

is such that

$$1/b(x) = \sum_{k=0}^{r-1} d_k x^k + O(x^r), \tag{17}$$

where

$$b(x) = \sum_{k=0}^{m-1} b_k x^k \text{ and } d_k = \frac{d^k}{dx^k}\left(\frac{1}{b(x)}\right)\bigg|_{x=0}. \tag{18}$$

The wrapped convolution satisfies the formula

$$a \otimes_m b = \left\{ F_m^{-1} \left[F_m(a) \cdot F_m(b) \right] + F_m^{-1} \left[F_m(\Psi \cdot a) \cdot F_m(\Psi \cdot b) \right] / \Psi \right\} / 2, \tag{19}$$

where $a = (a_0, a_1, \ldots, a_{m-1})$, $b = (b_0, b_1, \ldots, b_{m-1})$, $\Psi = (1, \psi_{2m}^1, \ldots, \psi_{2m}^{m-1})$, ψ_{2m} is the primitive root of order $2m$ of the unity in K, and vector operations of multiplication and division are defined coordinatewise. Formula (19) gives an extremely effective and fast algorithm of the order $O(m \log m)$ to evaluate wrapped convolutions, which is observed implicitly in [**7**], see also [**8**]. Note that it can be also applied to evaluate

$$b_j = \sum_{i=0}^{m-1} a_i \gamma^{ij} \quad (j = 1, 2, \ldots, m-1). \tag{20}$$

Indeed, we have

$$b_j = \sum_{i=0}^{m-1} a_i \gamma^{ij} = r_j \left(\sum_{i=0}^{j} p_i q_{j-i} + \sum_{i=j+1}^{m-1} p_i q_{-(j-i+1)} \right) \; (j = 0, 1, ..., m-1), \tag{21}$$

where

$$r_j = \prod_{k=0}^{j} \gamma^k, \quad p_j = a_j \prod_{k=0}^{j-1} \gamma^k, \quad q_j = \frac{1}{\prod_{k=0}^{j} \gamma^k} \quad (j = 0, 1, \ldots, m-1).$$

Hence

$$b_{j-(m-1)} = r_{j-(m-1)} \sum_{i=0}^{j} d_i c_{j-i} \quad (j = m-1, m, \ldots, 2m-2)$$

with

$$d_i = \begin{cases} p_i & \text{for} \quad i = 0, 1, ..., m-1, \\ 0 & \text{for} \quad i = m, m+1, ..., 2m-2 \end{cases}$$

and

$$c_i = \begin{cases} q_{m-2-i} & \text{for} \quad i = 0, 1, ..., m-2, \\ q_{i-(m-1)} & \text{for} \quad i = m-1, m, ..., 2m-2. \end{cases}$$

Consequently, if $d = (d_i)_{i=0}^{2m-2}$, $c = (c_i)_{i=0}^{2m-2}$ and $r = (r_i)_{i=0}^{m-1}$, then we get

$$b = \left(d \widetilde{\otimes}_m c \right) \cdot r, \tag{22}$$

where

$$d \widetilde{\otimes}_m c = P_m \left(d \otimes_{2m-1} c \right)$$

and the projection $P_m : K^{2m-1} \to K^m$ is defined by

$$P_m(e) = (e_{m-1}, e_m, \ldots, e_{2m-2}), \; e = (e_0, e_1, \ldots, e_{2m-2}). \tag{23}$$

It is clear that the order of algorithm (22) is equal to $O(m \log m)$. Note, that another algorithm for computing (20), which has the same order of complexity, was presented in [**10**].

The wrapped convolution can be also applied to evaluate the multidimensional convolution

$$u = a \otimes b, \tag{24}$$

of a hypermatrix $a = (a_\alpha)_{\alpha \in Q_n}$ and vector $b = (b_i)_{i=1}^d$, with $b_i = (b_{i,0}, b_{i,1}, \ldots, b_{i,n_i-1})$. Here coordinates of u are equal to

$$u_\alpha = \sum_{\beta \in Q_\alpha} a_\beta b_{\alpha-\beta}, \ \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d) \in Q_n. \tag{25}$$

Definition 1 ([**2**]). A hypermatrix

$$w = (w_\alpha)_{\alpha \in Q_n} = a \otimes^{(i)} b_i \in K^{n_1 \times n_2 \times \cdots \times n_d}, \ 1 \le i \le d, \tag{26}$$

is said to be the i-th partial hypermatrix convolution of a hypermatrix $a = (a_\alpha)_{\alpha \in Q_n}$ and a vector $b_i = (b_{i,0}, b_{i,1}, \ldots, b_{i,n_i-1})$, whenever each column

$$w_{\beta_1,\ldots,\beta_{i-1},\bullet,\beta_{i+1},\ldots,\beta_d} = a_{\beta_1,\ldots,\beta_{i-1},\bullet,\beta_{i+1},\ldots,\beta_d} \otimes_{n_i} b_i, \ 0 \le \beta_j < n_{j-1},$$
$$j = 1, 2, \ldots, i-1, i+1, \ldots, d,$$

of the hypermatrix w is equal to the wrapped convolution of the column

$$a_{\beta_1,\ldots,\beta_{i-1},\bullet,\beta_{i+1},\ldots,\beta_d} = \left(a_{\beta_1,\ldots,\beta_{i-1},j,\beta_{i+1},\ldots,\beta_d}\right)_{j=0}^{n_i-1}. \tag{27}$$

and vector b_i.

The notation of the partial hypermatrix convolutions enables to rewrite the multidimensional convolution $u = (u_\alpha)_{\alpha \in Q_n}$ in the following hypermatrix form

$$u = a \otimes b = \left(\ldots\left(\left(a \otimes^{(1)} b_1\right) \otimes^{(2)} b_2\right) \otimes^{(3)} \ldots\right) \otimes^{(d)} b_d \tag{28}$$

with $b_i = (b_{i,0}, b_{i,1}, \ldots, b_{i,n_i-1})$ and $a = (a_\alpha)_{\alpha \in Q_n}$ [**2**]. Hence it is clear that the fast algorithm for computing an i-th partial hypermatrix convolution should evaluate N/n_i one-dimensional convolutions for vectors of size n_i. Therefore, algorithm (28) for computing the multidimensional convolution is of the order

$$(N/n_1)\, O(n_1 \log n_1) + \ldots + (N/n_d)\, O(n_d \log n_d) = O(N \log N), N = n_1 n_2 \ldots n_d. \tag{29}$$

The same order is in the algorithm

$$v = a \widetilde{\otimes} b = \left(\ldots\left(\left(a \widetilde{\otimes}^{(1)} b_1\right) \widetilde{\otimes}^{(2)} b_2\right) \widetilde{\otimes}^{(3)} \ldots\right) \widetilde{\otimes}^{(d)} b_d, \tag{30}$$

where the i-th extended convolution $a \widetilde{\otimes}^{(i)} b_i$ $i = 1, 2, \ldots, d$ is defined as in Definition 1 with

$$a_{\beta_1,\ldots,\beta_{i-1},\bullet,\beta_{i+1},\ldots,\beta_d} \otimes_{n_i} b_i \tag{31}$$

replaced by

$$P_{n_i}\left(a_{\beta_1,\ldots,\beta_{i-1},\bullet,\beta_{i+1},\ldots,\beta_d} \otimes_{2n_i-1} b_i\right) \tag{32}$$

and $a_\alpha = 0$ for $\alpha \notin Q_n$.

In a similar way one can define the hypermatrix deconvolution

$$a = u \oslash b, \tag{33}$$

whenever $b_{i,0} \neq 0$ for $i = 0, 1, \ldots, n_i - 1$. The only difference consists in replacing the operator $\otimes^{(i)}$ of the i -th partial hypermatrix convolution in Definition 1 by the corresponding operator $\oslash^{(i)}$ of the i-th partial hypermatrix deconvolution. In other words, each column of the hypermatrix

$$a = (a_\alpha)_{\alpha \in Q_n} = w \oslash^{(i)} b_i \in K^{n_1 \times n_2 \times \cdots \times n_d},\ 1 \leq i \leq d, \tag{34}$$

should be equal to

$$\begin{aligned} a_{\beta_1,\ldots,\beta_{i-1},\bullet,\beta_{i+1},\ldots,\beta_d} &= w_{\beta_1,\ldots,\beta_{i-1},\bullet,\beta_{i+1},\ldots,\beta_d} \oslash_{n_i} b_i \\ &= w_{\beta_1,\ldots,\beta_{i-1},\bullet,\beta_{i+1},\ldots,\beta_d} \otimes_{n_i} b_i^{-1}, \end{aligned}$$

where $0 \leq \beta_j < n_j$. Then we have

$$\begin{aligned} a &= \left(\left(\ldots \left(u \oslash^{(d)} b_d\right) \ldots\right) \oslash^{(2)} b_2\right) \oslash^{(1)} b_1 \\ &= \left(\left(\ldots \left(u \otimes^{(d)} b_d^{-1}\right) \ldots\right) \otimes^{(2)} b_2^{-1}\right) \otimes^{(1)} b_1^{-1}. \end{aligned} \tag{35}$$

One can prove that the last algorithm for hypermatrix deconvolution is of the order

$$O(N \log N), N = n_1 n_2 \ldots n_d. \tag{36}$$

For this purpose, it is sufficient to observe that the convolutionary inverse of a vector $b = (b_0, b_1, \ldots, b_{m-1}) \in K^m$ with $b_0 \neq 0$ can be computed by the Newton method of the order $O(m \log m)$ [**11**]. More precisely, let

$$x_{i+1} = 2x_i - x_i^2 b,\ i = 0, 1, \ldots, \tag{37}$$

be the Newton iterative formula for the function $f(x) = x^{-1} - b\ \ (x \neq 0)$. Moreover, suppose that the coefficients

$$d_0, d_1, \ldots, d_{2^i - 1}\ \ (i \geq 1) \tag{38}$$

of the inverse

$$\left(b_0 + b_1 x + \ldots b_{m-1} x^{m-1}\right)^{-1} = d_0 + d_1 x + \ldots d_{2^i-1} x^{2^i - 1} + O(x^{2^i}) \tag{39}$$

are already computed and that $d_k = 0$ for all $k \geq 2^i$. Then the single Newton iteration

$$d = 2 \cdot d - d \otimes_{2^{i+1}} d \otimes_{2^{i+1}} b.$$

doubles the number of evaluated coefficients d_k $\left(k = 0, 1, \ldots, 2^{i+1} - 1\right)$ of the convolutionary inverse. Hence we finally conclude that the iterative Newton formula

$$d = 2 \cdot d - d \otimes_{2^i} d \otimes_{2^i} b,\ i = 2, 3, \ldots, \lceil \log_2 m \rceil, \tag{40}$$

with the starting vector d of the form

$$d = \left(\frac{1}{b_0}, -\frac{b_1}{b_0^2}, 0, 0 \ldots\right),$$

generates the required convolutionary inverse

$$d = (d_0, d_1, \ldots, d_{m-1}) \tag{41}$$

of $b = (b_0, b_1, \ldots, b_{m-1})$, $b_0 \neq 0$. Since the computational complexity of the convolution $\otimes_{2^i}$ is equal to $O(i2^i)$, it is clear that the computational complexity of algorithm (40) is of the order

$$O\left(m \log_2 m + \frac{m}{2} \log_2 \frac{m}{2} + \ldots + 2 \log_2 2\right) = O\left(m \log m\right). \tag{42}$$

This completes the proof that the algorithm (35) is of the order $O\left(N \log N\right)$.

3 Fast Bernstein-Lagrange transformation

Now we establish a fast algorithm for evaluating the multivariate polynomial

$$p_n\left(x\right) = \sum_{\alpha \in Q_n} f_\alpha \binom{n-1}{\alpha} x^\alpha \left(1-x\right)^{n-\alpha-1} \tag{43}$$

at the points $x_\beta = (x_{1,\beta_1}, x_{2,\beta_2}, \ldots, x_{d,\beta_d})$ with the coordinates of the form

$$x_{i,j} = \lambda_i \gamma_i^j \quad (i = 1, 2, \ldots, d,\ j = 0, 1, \ldots, n_i - 1). \tag{44}$$

For this purpose, note that

$$\begin{aligned} p_n\left(x\right) &= \sum_{\alpha \in Q_n} f_\alpha \sum_{\beta \in Q_{n-\alpha}} \binom{n-1}{\alpha} \binom{n-\alpha-1}{\beta} x^{\alpha+\beta} \left(-1\right)^\beta \\ &= \sum_{\beta \in Q_n} x^\beta \sum_{\alpha \in Q_{\beta+1}} f_\alpha \binom{n-1}{\alpha} \binom{n-\alpha-1}{\beta-\alpha} \left(-1\right)^{\beta-\alpha} \\ &= \sum_{\beta \in Q_n} a_\beta x^\beta, \end{aligned} \tag{45}$$

where the coefficients a_β are given by the formula

$$a_\beta = \frac{(n-1)!}{(n-\beta-1)!} \sum_{\alpha \in Q_{\beta+1}} \frac{f_\alpha \left(-1\right)^{\beta-\alpha}}{\alpha! \left(\beta-\alpha\right)!}.$$

Hence one can use the multidimensional convolution in order to get the algorithm

$$a = \left(\frac{f}{r} \otimes p\right) \cdot t = \left(\ldots \left(\left(\frac{f}{r} \otimes^{(1)} p_1\right) \otimes^{(2)} p_2\right) \otimes^{(3)} \cdots \otimes^{(d)} p_d\right) \cdot t, \tag{46}$$

where $t = \left(t_\alpha\right)_{\alpha \in Q_n}, r = \left(r_\alpha\right)_{\alpha \in Q_n}$ and $p_i = (p_{i,0}, p_{i,1}, \ldots, p_{i,n_i-1})$ are defined by

$$r_\alpha = \alpha!, \quad t_\alpha = \frac{(n-1)!}{(n-\alpha-1)!}, \quad \alpha \in Q_n, \tag{47}$$

and

$$p_{i,l} = \frac{\left(-1\right)^l}{(l)!} \quad (i = 1, 2, ..., d,\ l = 0, 1, ..., n_i - 1). \tag{48}$$

Therefore, it follows from (28) and (29) that the coefficients a_β can be computed by the algorithm (46) of the order $O(N \log N)$. Furthermore, by inserting formula (44) into (45), we get

$$y_\alpha = p_n(x_\alpha) = \sum_{\beta \in Q_n} a_\beta \lambda^\beta \gamma^{\alpha\beta}.$$

Hence, we obtain

$$y_\alpha = \left(\sum_{\beta_d=0}^{n_d-1} \cdots \left(\sum_{\beta_2=0}^{n_2-1} \left(\sum_{\beta_1=0}^{n_1-1} a_\beta b_\beta w_{1,\alpha_1-\beta_1}\right) w_{2,\alpha_2-\beta_2}\right) \cdots w_{d,\alpha_d-\beta_d}\right) q_\alpha, \tag{49}$$

whenever we set

$$q_\beta = \prod_{j=1}^{d} \prod_{k=0}^{\beta_j} \gamma_j^k, \quad b_\beta = \prod_{j=1}^{d} \lambda_j^{\beta_j} \prod_{k=0}^{\beta_j-1} \gamma_j^k, \quad \beta \in Q_n, \tag{50}$$

and

$$w_{j,l} = \frac{1}{\prod_{k=0}^{l} \gamma_j^k} \quad (j = 1, 2, \ldots, d,\ l = 0, 1, \ldots, n_j - 1). \tag{51}$$

Finally, formula (49) yields the following theorem.

Theorem 1. If $\mathcal{T} : (f_\alpha)_{\alpha \in Q_n} \to (y_\alpha)_{\alpha \in Q_n}$ denotes the d-dimensional Bernstein-Lagrange transformation with the points $x_\alpha = (x_{1,\alpha_1}, x_{2,\alpha_2}, \ldots, x_{d,\alpha_d})$ defined as in (44), then $\mathcal{T}$ can be evaluated by the algorithm

$$\mathcal{T}: \ y = \left(\cdots\left(\left((a \cdot b) \widetilde{\otimes}^{(1)} w_1\right) \widetilde{\otimes}^{(2)} w_2\right) \widetilde{\otimes}^{(3)} \cdots \widetilde{\otimes}^{(d)} w_d\right) \cdot q, \tag{52}$$

$$a = \left(\cdots\left(\left(\frac{f}{r} \otimes^{(1)} p_1\right) \otimes^{(2)} p_2\right) \otimes^{(3)} \cdots \otimes^{(d)} p_d\right) \cdot t, \tag{53}$$

where elements of $b = (b_\alpha)_{\alpha \in Q_n}$, $q = (q_\alpha)_{\alpha \in Q_n}$, $w_i = (w_{i,n_i-2}, \ldots, w_{i,0}, w_{i,0}, w_{i,1}, \ldots, w_{i,n_i-1})$, $r = (r_\alpha)_{\alpha \in Q_n}$, $t = (t_\alpha)_{\alpha \in Q_n}$ and $p_i = (p_{i,n_i-2}, \ldots, p_{i,0}, p_{i,0}, w_{i,1}, \ldots, p_{i,n_i-1})$ are defined as in formulae (47), (48), (50) and (51). Moreover, this algorithm has the running time of $O(N \log(N)) + O(dN)$, where $N = n_1 n_2 \cdots n_d$.

We note that the term $O(dN)$ in the running time is an estimate of all auxiliary computations (47), (48), (50) and (51), which do not use convolutions. For the completeness of consideration, we summarize the algorithm for computing the multidimensional Bernstein-Lagrange transformation in more detail.

Algorithm 1. The d - dimensional Bernstein-Lagrange transformation $\mathcal{T}$ with respect to the points $x_\alpha = (x_{1,\alpha_1}, x_{2,\alpha_2}, \ldots, x_{d,\alpha_d})$, where $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d) \in Q_n$, $n = (n_1, n_2, \ldots, n_d)$ and $x_{i,j} = \lambda_i \gamma_i^j$ $(i = 1, 2, \ldots, d,\ j = 0, 1, 2, \ldots, n_i - 1)$.
Input: A hypermatrix $f = (f_\alpha)_{\alpha \in Q_n}$, scalar vectors $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_d)$ and $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_d)$ in K^d, and the vector $n = (n_1, n_2, \ldots, n_d)$ of positive integers.
Output: A hypermatrix $y = (y_\alpha)_{\alpha \in Q_n}$ of values $y_\alpha = p_n(x_\alpha)$.

1. Use (47) to evaluate r_α, t_α for each $\alpha \in Q_n$.
2. Use (48) to evaluate $p_{j,l}$ for $j = 1, 2, \ldots, d, l = 0, 1, \ldots, n_j - 1$.

3. Perform the componentwise division $v = f/r$.
4. For i from 1 to d do the following:
   4.1. Compute the partial hypermatrix convolution $v = v \otimes^{(i)} p_i$.
5. Perform the componentwise multiplication $a = v \cdot t$.
6. Use (50) to evaluate b_β, q_β for each $\beta \in Q_n$.
7. Use (51) to evaluate $w_{j,l}$ for $j = 1, 2, \ldots, d, l = 0, 1, \ldots, n_j - 1$.
8. Perform the componentwise multiplication $u = a \cdot b$.
9. For i from 1 to d do the following:
   9.1. Compute the extended partial hypermatrix convolution $u = u \widetilde{\otimes}^{(i)} w_i$.
10. Perform the componentwise multiplication $y = u \cdot q$.
11. Return (y).

4 Inverse multidimensional Bernstein-Lagrange transformation

Now we consider the inversion of the multidimensional Bernstein-Lagrange transformation

$$\mathcal{T}^{-1} : (y_\alpha)_{\alpha \in Q_n} \to (f_\alpha)_{\alpha \in Q_n}. \tag{54}$$

If we know coefficients $y_\alpha = p(x_\alpha)$ of the Lagrange polynomial (6) at the knots

$$x_\alpha = (x_{1,\alpha_1}, x_{2,\alpha_2}, \ldots, x_{d,\alpha_d}) \tag{55}$$

of the form

$$x_{i,j} = \lambda_i \gamma_i^j,\ i = 1, 2, .., d,\ j = 0, 1, \ldots, n_i - 1, \tag{56}$$

then we can find the multivariate divided differences

$$c_\alpha \quad = \quad p_n\left[x_{1,0}, \ldots, x_{1,\alpha_1}; \ldots; x_{d,0}, \ldots, x_{d,\alpha_d}\right] = \sum_{\beta \in Q_{\alpha+1}} \frac{y_\beta}{\prod\limits_{i=1}^{d} \prod\limits_{j=0, j\neq\beta_i}^{\alpha_i} (x_{i,\beta_i} - x_{i,j})} \tag{57}$$

of the Newton polynomial

$$p_n(x) = \sum_{\alpha \in Q_n} c_\alpha \prod_{i=1}^{d} \prod_{j=0}^{\alpha_i - 1} (x_i - x_{i,j}), \tag{58}$$

by using an algorithm of the order $O(N \log(N)) + O(dN)$ presented in [**12**]. Moreover, by using equality (56) the formula (58) can be rewritten in the following form

$$p_n(x) = \sum_{\alpha_1=0}^{n_1} \sum_{\alpha_2=0}^{n_2} \cdots \sum_{\alpha_d=0}^{n_d} c_{\alpha_1, \alpha_2, \ldots, \alpha_d} \prod_{i=1}^{d} x_i^{\alpha_i} \prod_{j=0}^{\alpha_i - 1} \left(1 - \frac{\lambda_i \gamma_i^j}{x_i}\right). \tag{59}$$

Since we have (see [**13**])

$$\prod_{k=0}^{n-1} \left(1 - xq^k\right) = \sum_{m=0}^{n} \begin{bmatrix} n \\ m \end{bmatrix}_q (-1)^m x^m q^{\frac{m(m-1)}{2}} \tag{60}$$

with

$$\begin{bmatrix} n \\ m \end{bmatrix}_q = \frac{\prod_{i=0}^{n}\left(1-q^i\right)}{\prod_{i=0}^{m}\left(1-q^i\right)\prod_{i=0}^{n-m}\left(1-q^i\right)} = \frac{[n]_q!}{[n-m]_q!\,[m]_q!},$$

it follows from (59) and (60) that

$$p_n(x) = \sum_{\beta_1=0}^{n_1-1}\sum_{\beta_2=0}^{n_2-1}\cdots\sum_{\beta_d=0}^{n_d-1} a_{\beta_1,\beta_2,\ldots,\beta_d}\prod_{i=1}^{d} x_i^{\beta_i},$$

where

$$a_\beta = \frac{1}{[\beta]_q!}\sum_{\alpha_1=0}^{n_1-\beta_1-1}\sum_{\alpha_2=0}^{n_2-\beta_2-1}\cdots\sum_{\alpha_d=0}^{n_d-\beta_d-1} c_{\alpha_1,\alpha_2,\ldots,\alpha_d}\prod_{i=1}^{d}\frac{[\alpha_i+\beta_i]_q!}{[\alpha_i]_q!}\gamma_i^{\frac{\alpha_i(\alpha_i-1)}{2}}\left(-\lambda_i\right)^{\alpha_i}. \tag{61}$$

Hence we get

$$a_\alpha = \frac{1}{[\beta]_q!}\sum_{\alpha\in Q_{n-\beta}} c_\alpha\frac{[\alpha+\beta]_q!}{[\alpha]_q!}\gamma^{\frac{\alpha(\alpha-1)}{2}}(-\lambda)^\alpha,\ \alpha\in Q_n,$$

or equivalently

$$a = \left(\ldots\left(\left(\left(c\cdot v\right)\overset{\leftarrow(d)}{\otimes} z_d\right)\overset{\leftarrow(d-1)}{\otimes} z_{d-1}\right)\overset{\leftarrow(d-2)}{\otimes}\cdots\overset{\leftarrow(1)}{\otimes} z_1\right)\cdot g, \tag{62}$$

where the elements of $v=(v_\alpha)$, $g=(g_\alpha)$ and $z_i=(z_{i,0},z_{i,1},\ldots,z_{i,n_i-1})$ are defined by

$$\begin{aligned} v_\alpha &= \prod_{i=1}^{d}\frac{1}{[\alpha_i]_q!}\gamma_i^{\frac{\alpha_i(\alpha_i-1)}{2}}\left(-\lambda_i\right)^{\alpha_i}, \quad g_\alpha = \frac{1}{[\alpha]_q!}, \quad \alpha\in Q_n, \\ z_{i,l} &= \prod_{i=1}^{d}[n_i-l]_q!\ \left(i=1,2,\ldots,d,\ l=0,1,\ldots,n_i-1\right), \end{aligned} \tag{63}$$

and the reversed i-th partial hypermatrix convolution

$$w\overset{\leftarrow(i)}{\otimes} z_i = \widehat{\widehat{w}\otimes^{(i)} z_i} \tag{64}$$

is defined as the i-th partial hypermatrix convolution with its elements written in the reverse order, where $\widehat{w}$ is the hypermatrix with i-th column written in the reverse order, too. Finally, one can apply (46) to get the following theorem.

Theorem 2. Let $\mathcal{T}^{-1}:(y_\alpha)_{\alpha\in Q_n}\to(f_\alpha)_{\alpha\in Q_n}$ be the inverse multidimensional Bernstein-Lagrange transformation with respect to the points $x_\alpha=(x_{1,\alpha_1},x_{2,\alpha_2},\ldots,x_{d,\alpha_d})$ with the coordinates of the form

$$x_{i,j}=\lambda_i\gamma_i^j,\ i=1,2,..,d,\ j=0,1,2,\ldots,n_i-1, \tag{65}$$

where $\lambda_i \neq 0$, $\gamma_i \neq 1$ and $\gamma_i \neq 0$. Then it can be evaluated by the algorithm

$$\mathcal{T}^{-1} : f \quad = \quad \left(\ldots \left(\left(\frac{a}{t} \oslash^{(d)} p_d \right) \oslash^{(d-1)} p_{d-1} \right) \ldots \right) \oslash^{(1)} p_1 \cdot r, \tag{66}$$

$$a \quad = \quad \left(\ldots \left(\left((c \cdot v) \overset{\leftarrow(d)}{\otimes} z_d \right) \overset{\leftarrow(d-1)}{\otimes} z_{d-1} \right) \overset{\leftarrow(d-2)}{\otimes} \cdots \overset{\leftarrow(1)}{\otimes} z_1 \right) \cdot g, \tag{67}$$

where the elements of $t = (t_\alpha)_{\alpha \in Q_n}$, $r = (r_\alpha)_{\alpha \in Q_n}$, $g = (g_\alpha)_{\alpha \in Q_n}$, $c = (c_\alpha)_{\alpha \in Q_n}$, $v = (v_\alpha)_{\alpha \in Q_n}$, $z_i = (z_{i,0}, z_{i,1}, \ldots, z_{i,n_i-1})$ and $p_i = (p_{i,0}, p_{i,1}, \ldots, p_{i,n_i-1})$ are defined as in formulae (47), (48), (57) and (63). Moreover, this algorithm has the running time of $O(N \log(N)) + O(dN)$, where $N = n_1 n_2 \cdots n_d$.

Algorithm 2. The inverse d-dimensional Bernstein-Lagrange transformation $\mathcal{T}^{-1}$ with respect to the points $x_\alpha = (x_{1,\alpha_1}, x_{2,\alpha_2}, \ldots, x_{d,\alpha_d})$, where $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d) \in Q_n$, $n = (n_1, n_2, \ldots, n_d)$ and $x_{i,j} = \lambda_i \gamma_i^j$ $(i = 1, 2, \ldots, d,\ j = 0, 1, 2, \ldots, n_i - 1)$.
Input: A hypermatrix $y = (y_\alpha)_{\alpha \in Q_n}$, scalar vectors $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_d)$, $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_d)$ in K^d, and the vector $n = (n_1, n_2, \ldots, n_d)$ of positive integers.
Output: A hypermatrix $f = (f_\alpha)_{\alpha \in Q_n}$.

1. Use Algorithm 8 from [**12**] to evaluate c_α for each $\alpha \in Q_n$.
2. Use (63) to evaluate v_α, g_α for each $\alpha \in Q_n$.
3. Use (63) to evaluate $z_{j,l}$ for $j = 1, 2, \ldots, d, l = 0, 1, \ldots, n_j - 1$.
4. Perform the componentwise multiplication $v = c \cdot v$.
5. For i from d down to 1 do the following:
   5.1. Compute the reversed partial hypermatrix convolution $v = v \overset{\leftarrow(i)}{\otimes} z_i$.
6. Perform the componentwise multiplication $a = v \cdot g$.
7. Use (47) to evaluate t_α, r_α for each $\alpha \in Q_n$.
8. Use (48) to evaluate $p_{j,l}$ for $j = 1, 2, \ldots, d, l = 0, 1, \ldots, n_j - 1$.
9. Perform the componentwise division $a = a/t$.
10. For i from d down to 1 do the following:
    10.1. Compute the partial hypermatrix deconvolution $a = a \oslash^{(i)} p_i$.
11. Perform the componentwise multiplication $f = a \cdot r$.
12. Return (f).

5 Conclusions and remarks

In this paper, we present two new algorithms for the d-dimensional Bernstein-Lagrange transformation and its inverse for the points

$$x_\alpha = (x_{1,\alpha_1}, x_{2,\alpha_2}, \ldots, x_{d,\alpha_d}),\ \alpha \in Q_n \tag{68}$$

with the coordinates defined by the formulae

$$x_{i,j} = \lambda_i \gamma_i^j,\ i = 1, 2, \ldots, d,\ j = 0, 1, \ldots, n_i - 1,$$

where $\gamma_i \neq 0$, $\gamma_i \neq 1$ and $\lambda_i \neq 0$ are fixed.

Roughly speaking, the main feature of these algorithms consists in splitting the computations into two steps. In the first step we compute only quantities, which require to perform only $O(dN)$ operations. The second step includes computations of d-dimensional convolutions or deconvolutions of the order $O(N \log N)$. Thus, the computational complexity of this algorithms takes only

$$O(N \log N) + O(dN) \tag{69}$$

operations, where $N = n_1 n_2 \dots n_d$. Moreover, if we make natural assumption that $n_i \geq 2$ for $i = 1, 2, \dots, d$, then $\log_2 N \geq d$ and the order of the algorithm can be reduced to $O(N \log(N))$.

It should be emphasized, that parts (53) and (66) of the algorithms presented in Theorems 1 and 2 are valid for arbitrary points $x_\alpha, \alpha \in Q_n$. However, we do not know if the remaining parts of these algorithms are true for the points defined in (11).

References

[1] Stoer J., Bulirsch R., Introduction to Numerical Analysis, Springer - Verlag, New York 1993.

[2] Kapusta J., Smarzewski R., Fast algorithms for multivariate interpolation and evaluation at special points, Journal of Complexity 25 (2009): 332.

[3] Farouki R., Rajan V. T., Algorithms for polynomials in Bernstein form, Computer Aided Geometric Design 5 (1988): 1.

[4] Mainara E., Peña J. M., Evaluation algorithms for multivariate polynomials in Bernstein-Bézier form, Journal of Approximation Theory 143 (1) (2006): 44.

[5] Peters J., Evaluation and approximate evaluation of the multivariate Bernstein-Bézier form on a regularly partitioned simplex, ACM Transactions on Mathematical Software 20(4) (1994): 460.

[6] Phien H. N., Dejdumrong N., Efficient algorithms for Bé zier curves, Computer Aided Geometric Design 17 (2000): 247.

[7] Aho A., Hopcroft J., Ullman J., The design and analysis of computer algorithms, Addison-Wesley, London 1974.

[8] Smarzewski R., Kapusta J., Fast Lagrange-Newton transformations, Journal of Complexity 23 (2007): 336.

[9] Bini D., Pan V. Y., Polynomial and matrix computations: fundamental algorithms, Birkhäuser Verlag, 1994.

[10] Aho A., Steiglitz K., Ullman J., Evaluating polynomials at fixed sets of points, SIAM Journal Comput. 4 (1975): 533.

[11] Borwein J. M., Borwein P. B., Pi and the AGM: A study in analytic number theory and computational complexity, Canadian Mathematical Society Series of Monographs and Advanced Texts, John Wiley and Sons, New York, Chichester, Brisbane, Toronto, Singapore, 1987.

[12] Kapusta J., An efficient algorithm for multivariate Maclaurin-Newton transformation, Annales UMCS Informatica AI VIII (2) (2008): 5.

[13] Andrews G. E., The theory of partitions (Encyclopedia of mathematics and its applications), Addison-Wesley Publishing Company, 1976.