Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational and Applied Mathematics 219 (2008) 156–169
JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS
www.elsevier.com/locate/cam

Error analysis of efficient evaluation algorithms for tensor product surfaces☆

J. Delgado[a,*], J.M. Peña[b]

[a]*Departamento de Matemáticas, Universidad de Oviedo, Spain*
[b]*Departamento de Matemática Aplicada, Universidad de Zaragoza, Spain*

Received 1 December 2006; received in revised form 8 June 2007

Abstract

Backward stability of the Casteljau algorithm and two more efficient algorithms for polynomial tensor product surfaces with interest in CAGD is shown. The conditioning of the corresponding bases are compared. These algorithms are also compared with the corresponding Horner algorithm and their higher accuracy is shown. A running error analysis of the algorithms is also carried out providing algorithms which calculate "a posteriori" sharp error bounds simultaneously to the evaluation of the surface without increasing significantly the computational cost.

MSC: 65D17; 65G50

Keywords: Tensor product surfaces; Evaluation algorithms; Corner cutting algorithms; Error analysis

1. Introduction

Horner algorithm is the most frequently used algorithm for polynomial evaluation. However, in the field of computer aided geometric design (CAGD) other algorithms are used for polynomial evaluation. One of the goals of this paper is to show the higher accuracy of these algorithms over Horner algorithm for the evaluation of tensor product surfaces. The most usual algorithm in CAGD for the evaluation of polynomial curves is the de Casteljau algorithm. This algorithm has quadratic time complexity when evaluating a polynomial curve of degree n, that is, of $\mathcal{O}(n^2)$ elementary operations. In the last two decades there has been an intense search of new algorithms in CAGD for the evaluation of polynomial curves more efficient than the de Casteljau algorithm (see [1–5,9,15,16]). In fact, there are other evaluation algorithms useful in design whose computational cost to evaluate a polynomial curve of degree n is linear, that is, of $\mathcal{O}(n)$ elementary operations. The computational cost becomes much more important when evaluating tensor product surfaces. We show in this paper that these algorithms present better stability properties than Horner algorithm. This opens the possibility of using them even outside of the field of CAGD.

Corner cutting algorithms play a key role in CAGD (see [12,8, Section 14.3, pp. 352–353]). In addition to the de Casteljau tensor product algorithm, we shall consider in Section 2 two more efficient alternative corner cutting

☆ Partially supported by MTM2006-03388 Research Grant, Spain.
* Corresponding author.
E-mail address: delgadojorge@uniovi.es (J. Delgado).

0377-0427/$ - see front matter
doi:10.1016/j.cam.2007.07.020

evaluation algorithms for tensor product surfaces derived from two evaluation algorithms for polynomial curves with linear complexity: the Wang–Ball algorithm (see [15]) and the evaluation algorithm introduced in [3] and adapted for the evaluation of surfaces in [9]. One advantage of this last algorithm over the Wang–Ball algorithm comes from the fact that the corresponding representation preserves the shape properties of the control polygon because it was proved in [3] that the corresponding basis is normalized totally positive (NTP), and it is well known that shape preserving representations are associated to NTP bases (see [13]). In contrast, in [5] it has been proved that the Wang–Ball basis is not NTP, although it satisfies the weaker property of monotonicity preservation.

As far as we know, in the literature there is no error analysis of the mentioned tensor product algorithms, even for the case of the de Casteljau algorithm. In Section 3 a backward and forward error analysis of the corner cutting algorithms for tensor product surfaces is performed. It is shown that they are backward stable and we also compare the conditioning of the bases. In Section 4, we carry out a running error analysis of the corner cutting algorithms, providing new algorithms which calculate "a posteriori" error bounds simultaneously to the evaluation of the surface and without increasing significantly the computational cost. The sharpness of these error bounds is shown in Section 5, which contains numerical experiments comparing the three algorithms considered in the paper and, in addition, the extension of the Horner algorithm for the evaluation of tensor product surfaces. Horner algorithm presents worse stability properties than the other algorithms. The conclusions are included at the end of Section 5.

2. Corner cutting evaluation algorithms for tensor product surfaces

We start this section by recalling the definition of tensor product surface.

Definition 1. Given two systems $U=(u_0,\dots,u_m)$ and $\overline{U}=(\overline{u}_0,\dots,\overline{u}_n)$ defined on $[a_1,b_1]$ and $[a_2,b_2]$, respectively, the system $U\otimes\overline{U}:=(u_i(x)\otimes\overline{u}_j(y))_{i=0,\dots,m}^{j=0,\dots,n}$ is called a tensor product system and the surface

$$F(x,y)=\sum_{i=0}^{m}\sum_{j=0}^{n}P_{ij}u_i(x)\overline{u}_j(y),\qquad (x,y)\in[a_1,b_1]\times[a_2,b_2], \tag{1}$$

is called a tensor product surface. The points P_{ij} $(i=0,\dots,m,\ j=0,\dots,n)$ are called control points of the surface F and the matrix $\{P_{ij}\}_{0\leqslant i\leqslant m;0\leqslant j\leqslant n}$ is called control net of F.

In [4] corner cutting systems were considered and it was proved that they always satisfy monotonicity preservation. Now let us generalize the concept of corner cutting system to the case of tensor product systems. Let us denote by $\Lambda_i(x)$ the $i\times(i+1)$ bidiagonal matrix

$$\Lambda_i(x):=\begin{pmatrix}1-\lambda_0^{(i)}(x) & \lambda_0^{(i)}(x) & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1-\lambda_{i-1}^{(i)}(x) & \lambda_{i-1}^{(i)}(x)\end{pmatrix}, \tag{2}$$

for all $i\in\{1,\dots,m\}$, where $\lambda_j^{(i)}:[a_1,b_1]\to[0,1]$ for $0\leqslant j<i\leqslant m$.

Definition 2. We say that $\Lambda_1(x)\cdots\Lambda_m(x)\otimes\overline{\Lambda}_n(y)^{\mathrm{T}}\cdots\overline{\Lambda}_1(y)^{\mathrm{T}}$ is a *tensor product corner cutting representation* on $[a_1,b_1]\times[a_2,b_2]$ of the tensor product system of functions $(u_i\otimes\overline{u}_j)_{0\leqslant i\leqslant m}^{0\leqslant j\leqslant n}$ defined on the set $[a_1,b_1]\times[a_2,b_2]$ if $U:=(u_0(x),\dots,u_m(x))=\Lambda_1(x)\cdots\Lambda_m(x)$ and $\overline{U}:=(\overline{u}_0(y),\dots,\overline{u}_n(y))=\overline{\Lambda}_1(y)\cdots\overline{\Lambda}_n(y)$, where the matrices $\Lambda_i(x)$ and $\overline{\Lambda}_i(y)$ are of the form (2), and the functions $\lambda_j^{(i)}:[a_1,b_1]\to[0,1]$ and $\overline{\lambda}_j^{(i)}:[a_2,b_2]\to[0,1]$ are continuous and increasing for all $0\leqslant j<i\leqslant m$ and $0\leqslant j<i\leqslant n$, respectively. Then we say that $U\otimes\overline{U}$ is a tensor product corner cutting system on $[a_1,b_1]\times[a_2,b_2]$.

By considering the components of the points P_{ij}, the evaluation of (1) depends on the evaluation of scalar functions. A tensor product corner cutting representation provides a corner cutting evaluation algorithm for tensor product surfaces, as the following algorithm shows.

Algorithm 3. *Let us consider the following tensor product surface represented in a tensor product system*

$$F(x,y)=\sum_{i=0}^{m}\sum_{j=0}^{n} f_{ij}u_i(x)\overline{u}_j(y). \tag{3}$$

Let us suppose that the system is a tensor product corner cutting system. Therefore, we can write (3) *in the following way*:

$$F(x,y)=\Lambda_1(x)\cdots\Lambda_m(x)(f_{ij})_{0\leqslant i\leqslant m}^{0\leqslant j\leqslant n}\overline{\Lambda}_n(y)^{\mathrm{T}}\cdots\overline{\Lambda}_1(y)^{\mathrm{T}}.$$

Then performing

1. $f_{ij}^{00}(x,y)=f_{ij}$ *for* $0\leqslant i\leqslant m$ *and* $0\leqslant j\leqslant n$.
2. $f_{ij}^{0s}(x,y)=(1-\overline{\lambda}_j^{(n+1-s)}(y))f_{ij}^{0,s-1}(x,y)+\overline{\lambda}_j^{(n+1-s)}(y)f_{i,j+1}^{0,s-1}(x,y)$ *for* $0\leqslant i\leqslant m$, $1\leqslant s\leqslant n$ *and* $0\leqslant j\leqslant n-s$.
3. $f_{i0}^{rn}(x,y)=(1-\lambda_i^{(m+1-r)}(x))f_{i0}^{r-1,n}(x,y)+\lambda_i^{(m+1-r)}(x)f_{i+1,0}^{r-1,n}(x,y)$ *for* $1\leqslant r\leqslant m$ *and* $0\leqslant i\leqslant m-r$, *we have* $f_{00}^{mn}(x,y)=F(x,y)$.

Remark 4. We can check that the previous algorithm uses $(\frac{1}{2})(m+1)n(n+1)+(\frac{1}{2})m(m+1)$ sums and $(m+1)n(n+1)+m(m+1)$ products. If $m<n$, by evaluating first the curves $f_j(x)=\sum_{i=0}^{m}f_{ij}u_i(x)$, for $j=0,\ldots,n$, and finally the curve $f(y)=\sum_{j=0}^{n}f_j\overline{u}_j(y)$, we can deduce an evaluation algorithm of the same kind that the previous one but with a lower computational cost. Then, if $m<n$, we will use this new algorithm and, if $m\geqslant n$, we will use Algorithm 3. So, from now on we will assume without loss of generality that $m\geqslant n$. See [8, p. 262] and [10].

In CAGD the usual representation of a tensor product polynomial surface is the Bernstein–Bézier form (see [6]) given by (3) and using the Bernstein tensor product basis $(b_i^m\otimes b_j^n)_{0\leqslant i\leqslant m}^{0\leqslant j\leqslant n}$, where $b_i^k(x)=\binom{k}{i}x^k(1-x)^{k-i}$, $i=0,1\ldots,k$, are the Bernstein polynomials. The corresponding surface is called Bézier surface (see [6]). The points P_{ij} ($i=0,\ldots,m$, $j=0,\ldots,n$) are called control points of the surface F and the matrix $\{P_{ij}\}_{0\leqslant i\leqslant m;0\leqslant j\leqslant n}$ is called Bézier control net of F.

Taking into account that the Bernstein tensor product basis admits the tensor product corner cutting representation given by

$$\lambda_j^{(i)}(x)=x,\quad 0\leqslant j<i\leqslant m\quad\text{and}\quad \overline{\lambda}_j^{(i)}(y)=y,\quad 0\leqslant j<i\leqslant n, \tag{4}$$

we can conclude that it is a tensor product corner cutting system. So, we can apply Algorithm 3 to the particular case where $U=(b_0^m,\ldots,b_m^m)=\Lambda_1\cdots\Lambda_m$, $\overline{U}=(b_0^n,\ldots,b_n^n)=\overline{\Lambda}_1\cdots\overline{\Lambda}_n$ with the functions $\lambda_j^{(i)}$ and $\overline{\lambda}_j^{(i)}$ given by (4), obtaining a corner cutting algorithm for the evaluation of Bézier surfaces, which will be called the *de Casteljau tensor product algorithm*. We can easily check that this algorithm consists of $(\frac{1}{2})(m+1)n(n+1)+(\frac{1}{2})m(m+1)$ sums and $(m+1)n(n+1)+m(m+1)$ products.

The Wang–Ball basis was presented by Wang in [16]. Let us recall the definition of the Wang–Ball basis $(a_0^m(t),\ldots,a_m^m(t))$, $m\geqslant 2$, $t\in[0,1]$:

$$a_i^m(t)=2^it^i(1-t)^{i+2},\quad 0\leqslant i\leqslant\left\lfloor\frac{m}{2}\right\rfloor-1,$$

$$a_i^m(t)=2^{m-i}t^{m+2-i}(1-t)^{m-i},\quad \left\lfloor\frac{m+1}{2}\right\rfloor+1\leqslant i\leqslant m.$$

In addition, if m is even, $a^m_{m/2}(t) = 2^{m/2}t^{m/2}(1-t)^{m/2}$, and, if m is odd,

$$a_{(m-1)/2}(t) = 2^{(m-1)/2}t^{(m-1)/2}(1-t)^{(m+1)/2}, \quad a_{(m+1)/2}(t) = 2^{(m-1)/2}t^{(m+1)/2}(1-t)^{(m-1)/2},$$

where $\lfloor r \rfloor$ $(r > 0)$ is the greatest positive integer less than or equal to r.

In [1] it was shown that this basis has an evaluation algorithm of linear complexity. In [5] we proved that the Wang–Ball basis is not NTP in general. Let us recall that the *collocation matrix* of a system of univariate functions $(u_0(t), \ldots, u_n(t))$ at $t_0 < \cdots < t_m$ is given by

$$M\begin{pmatrix} u_0, \ldots, u_n \\ t_0, \ldots, t_m \end{pmatrix} := (u_j(t_i))_{i=0,\ldots,m;\, j=0,\ldots,n}, \tag{5}$$

and hence its (i, j)-entry is $u_j(t_i)$. A system $(u_0, \ldots, u_n)$ is normalized if $\sum_{i=0}^n u_i(t) = 1$ for all t. A matrix is *totally positive* (*TP*) if all its minors are nonnegative and a system of functions is TP when all its collocation matrices (5) are TP. In case of a normalized totally positive (NTP) basis one knows that the curve imitates the shape of its control polygon, due to the variation diminishing properties of TP matrices (see [13]). So, since the Wang–Ball basis is not NTP, we cannot hope that it satisfies many shape preserving properties. Nevertheless, we also saw in [5] that the Wang–Ball basis is monotonicity preserving.

Taking into account the evaluation algorithm given in [15] for Wang–Ball curves and considering in Algorithm 3 the particular case where the systems $U = (a^m_0, \ldots, a^m_m) = \Lambda_1 \cdots \Lambda_m$ and $\overline{U} = (a^n_0, \ldots, a^n_n) = \overline{\Lambda}_1 \cdots \overline{\Lambda}_n$ are Wang–Ball bases with

$$\lambda^{(i)}_j(x) = \overline{\lambda}^{(i)}_j(y) = 0 \quad \text{for } j \in \left\{0, \ldots, \left\lfloor \frac{i-3}{2} \right\rfloor\right\}, \tag{6}$$

$$\lambda^{(i)}_j(x) = x, \ \overline{\lambda}^{(i)}_j(y) = y \quad \text{for } j \in \left\{\left\lfloor \frac{i-1}{2} \right\rfloor, \left\lfloor \frac{i}{2} \right\rfloor\right\}, \tag{7}$$

$$\lambda^{(i)}_j(x) = \overline{\lambda}^{(i)}_j(y) = 1 \quad \text{for } j \in \left\{\left\lfloor \frac{i}{2} + 1 \right\rfloor, \ldots, i-1\right\}, \tag{8}$$

we can deduce an algorithm for the evaluation of the Wang–Ball tensor product surfaces. The corresponding basis $U \otimes \overline{U}$ will be called *Wang–Ball tensor product basis* and the algorithm will be called *Wang–Ball tensor product algorithm*. We can easily check that this algorithm consists of $(\frac{1}{2})((m+1)(3n-1)+(3m-1))$ sums and $(m+1)(3n-1)+(3m-1)$ products if n and m are odd, of $(\frac{1}{2})((m+1)(3n-1)+3m)$ sums and $(m+1)(3n-1)+3m$ products if n is odd and m is even, of $(\frac{1}{2})((m+1)3n+(3m-1))$ sums and $(m+1)3n+(3m-1)$ products if n is even and m is odd, and of $(\frac{1}{2})((m+1)3n+3m)$ sums and $(m+1)3n+3m$ products if n and m are even. In conclusion, the Wang–Ball tensor product algorithm has a computational cost of $\mathcal{O}(mn)$ elementary operations in contrast to the cost of $\mathcal{O}(mn^2)$ elementary operations of the de Casteljau tensor product algorithm.

In [3] Delgado and Peña introduced another basis $(c^m_0(t), \ldots, c^m_m(t))$, $m \geqslant 2$, $t \in [0, 1]$, with a corner cutting evaluation algorithm associated of linear complexity:

$$c^m_0(t) = (1-t)^m, \quad c^m_m(t) = t^m,$$

$$c^m_i(t) = t(1-t)^{m-i}, \quad 1 \leqslant i \leqslant \left\lfloor \frac{m}{2} \right\rfloor - 1,$$

$$c^m_i(t) = t^i(1-t), \quad \left\lfloor \frac{m+1}{2} \right\rfloor + 1 \leqslant i \leqslant m-1.$$

In addition, if m is even, $c^m_{m/2}(t) = 1 - t^{m/2+1} - (1-t)^{m/2+1}$, and, if m is odd, $c^m_{(m-1)/2}(t) = t(1-t)^{(m+1)/2} + f_m(t)$ and $c^m_{(m+1)/2}(t) = f_m(t) + t^{(m+1)/2}(1-t)$ where $f_m(t) = \frac{1}{2}[1 - t^{(m+1)/2} - (1-t)^{(m+1)/2}]$.

In [3] it was proved that this basis presents nice shape preserving properties because it is NTP. In addition it was also shown that it is associated to a corner cutting evaluation algorithm of linear complexity. Following the notation in [9], we call it *DP basis*.

Taking into account the evaluation algorithm given in [3] and taking in Algorithm 3 $U = (c_0^m, \ldots, c_m^m) = \Lambda_1 \cdots \Lambda_m$ and $\overline{U} = (c_0^n, \ldots, c_n^n) = \overline{\Lambda}_1 \cdots \overline{\Lambda}_n$ with

$$\lambda_0^{(i)}(x) = \lambda_{i-1}^{(i)}(x) = x, \quad \overline{\lambda}_0^{(i)}(y) = \overline{\lambda}_{i-1}^{(i)}(y) = y, \tag{9}$$

$$\lambda_j^{(i)}(x) = \overline{\lambda}_j^{(i)}(y) = 1, \quad j = 1, \ldots, \left\lfloor \frac{i}{2} \right\rfloor - 1, \tag{10}$$

$$\lambda_j^{(i)}(x) = \overline{\lambda}_j^{(i)}(y) = 0, \quad j = \left\lfloor \frac{i+1}{2} \right\rfloor, \ldots, i-2, \tag{11}$$

$$\lambda_{(i-1)/2}^{(i)}(x) = \overline{\lambda}_{(i-1)/2}^{(i)}(y) = \tfrac{1}{2} \quad \text{if } i \text{ is odd}, \tag{12}$$

we deduce an algorithm for evaluating the corresponding linear NTP tensor product surfaces. We call the associated basis *DP tensor product basis* and to the algorithm *DP tensor product algorithm*. We can easily check that this algorithm consists of $2(m+1)n + 2m$ sums and $4(m+1)n + 4m$ products if n and m are odd, of $2(m+1)n + (2m-1)$ sums and $4(m+1)n + (4m-2)$ products if n is odd and m is even, of $(m+1)(2n-1) + 2m$ sums and $(m+1)(4n-2) + 4m$ products if n is even and m is odd, and of $(m+1)(2n-1) + (2m-1)$ sums and $(m+1)(4n-2) + (4m-2)$ products if n and m are even. In conclusion, the DP tensor product algorithm has a computational cost of mn order, in contrast to the mn^2 order of the de Casteljau tensor product algorithm.

3. Error analysis

In this section we carry out the error analysis of the corner cutting algorithms for the evaluation of tensor product surfaces (Algorithm 3) so that we can apply it to the particular cases of Bézier, Wang–Ball and DP tensor product surfaces. We also study the relation between the condition numbers of the associated bases.

Let us now introduce some standard notations in error analysis. Given $a \in \mathbb{R}$, the computed element in floating point arithmetic will be denoted by either $\mathrm{fl}(a)$ or by $\hat{a}$. As usual, to investigate the effect of rounding errors when working with floating point arithmetic we use either the model

$$\mathrm{fl}(a \text{ op } b) = (a \text{ op } b)(1+\delta), \quad |\delta| \leqslant u, \tag{13}$$

or the model

$$\mathrm{fl}(a \text{ op } b) = \frac{a \text{ op } b}{1+\delta}, \quad |\delta| \leqslant u, \tag{14}$$

with u the unit roundoff and *op* any of the elementary operations $+, -, \times, /$. Given $k \in \mathbb{N}_0$ such that $ku < 1$, let us define

$$\gamma_k := \frac{ku}{1-ku}. \tag{15}$$

In our error analysis we shall deal with quantities satisfying that their absolute value is bounded above by γ_k. We denote such quantities by θ_k.

In order to perform the error analysis of an algorithm, the conditioning of the corresponding problem is also a very important aspect that must be taken into account. Given $F(x, y) = \sum_{i=0}^m \sum_{j=0}^n c_{ij} u_i(x) \overline{u}_j(y)$, where $u^{mn} = (u_i \otimes \overline{u}_j)_{0 \leqslant i \leqslant m}^{0 \leqslant j \leqslant n}$ is a basis of the corresponding space,

$$S_{u^{mn}}(F(x, y)) := \sum_{i=0}^m \sum_{j=0}^n |c_{ij} u_i(x) \overline{u}_j(y)|, \tag{16}$$

is called a condition number for the evaluation of $F(x, y)$ with the basis u^{mn} (see [14]).

The following result performs the backward and the forward error analysis of the corner cutting algorithms for the evaluation of tensor product surfaces.

Theorem 5. *Let us consider a tensor product corner cutting system* $u^{mn} = (u_i \otimes \overline{u}_j)_{0\leqslant i\leqslant m}^{0\leqslant j\leqslant n}$ *defined on* $I \times \overline{I}$ *where* $I, \overline{I} \subseteq \mathbb{R}$. *Let* $F(x, y)$ *be given by* (3) *and let us suppose that* $2(m+n)u < 1$, *where* u *is the unit roundoff. Then the value* $\widehat{F}(x, y) = fl(F(x, y))$ *computed in floating point arithmetic through Algorithm* 3 *satisfies*

$$\widehat{F}(x, y) = \sum_{i=0}^{m}\sum_{j=0}^{n} \overline{f}_{ij} u_i(x)\overline{u}_j(y), \quad \overline{f}_{ij} = f_{ij}(1 + \theta_{2(m+n)}(i, j)), \tag{17}$$

for certain unknown quantities $\theta_{2(m+n)}(i, j)$ *such that* $|\theta_{2(m+n)}(i, j)| \leqslant \gamma_{2(m+n)}$. *The computed value* $\widehat{F}(x, y)$ *also verifies*

$$|F(x, y) - \widehat{F}(x, y)| \leqslant \gamma_{2(m+n)} S_{u^{mn}}(F(x, y)) \tag{18}$$

and

$$|F(x, y) - \widehat{F}(x, y)| \leqslant \gamma_{2(m+n)} \max_{0\leqslant i\leqslant m; 0\leqslant j\leqslant n} |f_{ij}|. \tag{19}$$

Proof. By Algorithm 3 we have that

$$f_{ij}^{0,n+1-k}(x, y) = (1 - \overline{\lambda}_j^{(k)}(y)) f_{ij}^{0,n-k}(x, y) + \overline{\lambda}_j^{(k)}(y) f_{i,j+1}^{0,n-k}(x, y)$$

for all $k \in \{1, \ldots, n\}$, $j \in \{0, 1, \ldots, k-1\}$ and $i \in \{0, 1, \ldots, m\}$. So, taking into account formula (13) we have

$$\widehat{f}_{ij}^{0,n+1-k}(x, y) = [(1-\overline{\lambda}_j^{(k)}(y))\widehat{f}_{ij}^{0,n-k}(x, y)(1+\delta_{ij}^{0,n-k}) + \overline{\lambda}_j^{(k)}(y)\widehat{f}_{i,j+1}^{0,n-k}(x, y)(1 + \delta_{i,j+1}^{0,n-k})](1 + \varepsilon_{ij}^{0,n+1-k}),$$

where $|\delta_{ij}^{0,n-k}|$, $|\delta_{i,j+1}^{0,n-k}|$ and $|\varepsilon_{ij}^{0,n+1-k}|$ are numbers less than or equal to the unit roundoff u. Then we can write

$$\widehat{f}_{ij}^{0,n+1-k}(x, y) = (1 - \overline{\lambda}_j^{(k)}(y))\widehat{f}_{ij}^{0,n-k}(x, y)(1 + \theta_2) + \overline{\lambda}_j^{(k)}(y)\widehat{f}_{i,j+1}^{0,n-k}(x, y)(1 + \theta_2)$$

for a certain unknown quantity θ_2 such that $|\theta_2| \leqslant \gamma_2$. Taking into account all the previous considerations we can easily prove by recurrence that

$$\widehat{f}_{i0}^{0n}(x, y) = \sum_{j=0}^{n} f_{ij}^{00}(1 + \theta_{2n})\overline{u}_j(y) \tag{20}$$

for all $i \in \{0, 1, \ldots, m\}$. Let us observe that in the previous formula the coefficients f_{ij}^{00} are exact since they come from the initialization performed in the step 1 of Algorithm 3. Analogously to the last formula we can prove

$$\widehat{F}(x, y) = \sum_{i=0}^{m} \widehat{f}_{i0}^{0n}(x, y)(1 + \theta_{2m})u_i(x). \tag{21}$$

So, from (20) and (21) we can deduce

$$\widehat{F}(x, y) = \sum_{i=0}^{m}\sum_{j=0}^{n} f_{ij}^{00}(1 + \theta_{2(m+n)})u_i(x)\overline{u}_j(y).$$

Then, by the last formula, taking into account that $f_{ij}^{00} = f_{ij}$ for all i and j, that $|\theta_k| \leqslant \gamma_k$ and denoting $\overline{f}_{ij} := f_{ij}(1 + \theta_{2(m+n)})$ for all $i \in \{0, 1, \ldots, m\}$ and $j \in \{0, 1, \ldots, n\}$, we derive (17). Finally, from (17), taking into account, again, that $|\theta_k| \leqslant \gamma_k$ and the definition of the condition number (see formula (16)) we have

$$\begin{aligned} |F(x, y) - \widehat{F}(x, y)| &= \left| \sum_{i=0}^{m}\sum_{j=0}^{n} f_{ij}\theta_{2(m+n)} u_i(x)\overline{u}_j(y) \right| \\ &\leqslant \gamma_{2(m+n)} \sum_{i=0}^{m}\sum_{j=0}^{n} |f_{ij}| u_i(x)\overline{u}_j(y) \end{aligned}$$

and (18) holds.

Obviously, (19) is a consequence of formula (18) since the functions $u_i(x)\overline{u}_j(y)$ are nonnegative and form a normalized system.

On one hand, taking into account that the tensor product Bernstein, Wang–Ball and DP bases are tensor product corner cutting systems, by formula (17) of Theorem 5 the corresponding corner cutting algorithms are backward stable. On the other hand, the forward error bound (18) depends on the condition number of the tensor product basis. Now we are going to compare the condition numbers of the tensor product bases associated to the different algorithms considered along the paper. The following result will play a key role for this purpose.

Proposition 6. *Let $U = (u_0, \ldots, u_m)$ and $V = (v_0, \ldots, v_m)$ be bases of the vector space of functions $\mathscr{U}$ and, let $\overline{U} = (\overline{u}_0, \ldots, \overline{u}_n)$ and $\overline{V} = (\overline{v}_0, \ldots, \overline{v}_n)$ be bases of another vector space of functions $\overline{\mathscr{U}}$. If M and $\overline{M}$ are the matrices such that*

$$U = VM \quad \text{and} \quad \overline{U} = \overline{V}\,\overline{M} \tag{22}$$

then

$$U \otimes \overline{U} = (V \otimes \overline{V})(M \otimes \overline{M}). \tag{23}$$

In addition, if M and $\overline{M}$ are nonnegative matrices then $M \otimes \overline{M}$ is also a nonnegative matrix.

Proof. Let K be the matrix relating the tensor product bases $U \otimes \overline{U}$ and $V \otimes \overline{V}$ such that

$$U \otimes \overline{U} = (V \otimes \overline{V})K. \tag{24}$$

Taking into account that V and $\overline{V}$ are linearly independent systems, we can deduce that there exist points $x_0 < \cdots < x_m$ and $y_0 < \cdots < y_n$ such that the collocation matrices

$$C_1 := M\begin{pmatrix} v_0, \ldots, v_m \\ x_0, \ldots, x_m \end{pmatrix} \quad \text{and} \quad C_2 := M\begin{pmatrix} \overline{v}_0, \ldots, \overline{v}_n \\ y_0, \ldots, y_n \end{pmatrix}$$

are nonsingular. Since the collocation matrix of the tensor product system $V \otimes \overline{V}$ is the Kronecker product of the corresponding collocation matrices of V and $\overline{V}$, $C_1 \otimes C_2$ is the collocation matrix of $V \otimes \overline{V}$ at the points

$$((x_i, y_j)_{j=0,\ldots,n})_{i=0,\ldots,m} := ((x_0, y_j)_{j=0,\ldots,n}, \ldots, (x_m, y_j)_{j=0,\ldots,n}).$$

Then, since the Kronecker product of nonsingular matrices is nonsingular (see [7, Corollary 4.2.11]), the collocation matrix $C_1 \otimes C_2$ is also nonsingular. Now let us also consider the collocation matrices

$$B_1 := M\begin{pmatrix} u_0, \ldots, u_m \\ x_0, \ldots, x_m \end{pmatrix}, \quad B_2 := M\begin{pmatrix} \overline{u}_0, \ldots, \overline{u}_n \\ y_0, \ldots, y_n \end{pmatrix}$$

and $B_1 \otimes B_2$ which, by the same previous reasoning, is the collocation matrix of $U \otimes \overline{U}$ at the points $((x_i, y_j)_{j=0,\ldots,n})_{i=0,\ldots,m}$. Then, taking collocation matrices at the points $((x_i, y_j)_{j=0,\ldots,n})_{i=0,\ldots,m}$ in formula (24) we derive

$$B_1 \otimes B_2 = (C_1 \otimes C_2)K. \tag{25}$$

Now, considering the collocation matrices at $x_0 < \cdots < x_m$ and $y_0 < \cdots < y_n$, respectively, in (22), we deduce that $B_1 = C_1 M$ and $B_2 = C_2\overline{M}$. So, since the Kronecker product of two products of matrices is the product of the corresponding Kronecker products (see [7, Lemma 4.2.10]), we have

$$B_1 \otimes B_2 = C_1 M \otimes C_2\overline{M} = (C_1 \otimes C_2)(M \otimes \overline{M}).$$

Then, by the previous formula and (25), we deduce

$$(C_1 \otimes C_2)K = (C_1 \otimes C_2)(M \otimes \overline{M}).$$

Finally, taking into account that $C_1 \otimes C_2$ is a nonsingular matrix, we can conclude that the matrix of change of basis can be written as $K = M \otimes \overline{M}$, and then (23) follows from (24). In addition, if M and $\overline{M}$ are nonnegative matrices,

taking into account the definition of Kronecker product of two matrices (see [7, Definition 4.2.1]), we can conclude that $M \otimes \overline{M}$ is also a nonnegative matrix.

In [11] it was proved that the Bernstein tensor product basis is optimally stable in the sense that there does not exist a basis of nonnegative functions of its space $\mathscr{U}$ which has smaller condition number for all $f \in \mathscr{U}$ and any point. The following result shows that, in fact, it is always better conditioned than the other two bases considered along the paper.

Theorem 7. *Let us consider the Bernstein tensor product basis $b^{mn} = (b_i^m \otimes b_j^n)_{0 \leqslant i \leqslant m}^{0 \leqslant j \leqslant n}$, the Wang–Ball tensor product basis $a^{mn} = (a_i^m \otimes a_j^n)_{0 \leqslant i \leqslant m}^{0 \leqslant j \leqslant n}$ and the DP tensor product basis $c^{mn} = (c_i^m \otimes c_j^n)_{0 \leqslant i \leqslant m}^{0 \leqslant j \leqslant n}$ of $\mathscr{U}$. Then we have*

$$S_{b^{mn}}(f(x, y)) \leqslant \min\{S_{a^{mn}}(f(x, y)), S_{c^{mn}}(f(x, y))\}$$

for all $f \in \mathscr{U}$ and points $(x, y) \in [0, 1] \times [0, 1]$.

Proof. By Theorems 5 and 6 of [3] the matrix A_k such that $(c_0^k, \ldots, c_k^k) = (b_0^k, \ldots, b_k^k)A_k$ is totally positive and, in particular, nonnegative. On the other hand, in [15, pp. 130–131] it was shown that the matrix B_k such that $(a_0^k, \ldots, a_k^k) = (b_0^k, \ldots, b_k^k)B_k$ is also nonnegative. Then, by Proposition 6, the corresponding Kronecker product matrix relating both previous tensor product basis with the Bernstein tensor product basis is nonnegative. So the result follows from [11, Lemma 3.1]. □

Horner algorithm, associated to the monomial basis $(1, x, \ldots, x^n)$, is not a corner cutting algorithm but it will be compared in Section 5 with the other algorithms of this paper. Taking into account that the matrix C_k such that $(1, x, \ldots, x^k) = (b_0^k, \ldots, b_k^k)C_k$ is nonnegative (see [6, p. 42]), we can also derive from Proposition 6 and Lemma 3.1 of [11] that the Bernstein tensor product basis is better conditioned than the tensor product monomial basis.

4. Running error analysis of tensor product corner cutting algorithms

The "a priori" bounds deduced in Theorem 5 are computed before performing the corresponding evaluation algorithms. But, in practical computations, it is also desirable to obtain a bound of the produced absolute error simultaneously to the evaluation, i.e. an "a posteriori" bound (also called running error). In the following result we present a running error analysis of Algorithm 3.

Theorem 8. *Given Algorithm 3 for evaluating (3), we have, for all $r \in \{1, \ldots, n\}$, $j \in \{0, \ldots, n-r\}$ and $i \in \{0, \ldots, m\}$,*

$$|\widehat{f}_{ij}^{\,0r}(x, y) - f_{ij}^{0r}(x, y)| \leqslant u\pi_{ij}^{0r},$$

where u is the unit roundoff, $\pi_{ij}^{00} = 0$ for all $i \in \{0, 1, \ldots, m\}$ and $j \in \{0, 1, \ldots, n\}$, and

$$\pi_{ij}^{0s} = (1 - \overline{\lambda}_j^{(n+1-s)}(y))\pi_{ij}^{0,s-1} + \overline{\lambda}_j^{(n+1-s)}(y)\pi_{i,j+1}^{0,s-1} + (1 - \overline{\lambda}_j^{(n+1-s)}(y))|\widehat{f}_{ij}^{\,0,s-1}(x, y)|$$
$$+ \overline{\lambda}_j^{(n+1-s)}(y)|\widehat{f}_{i,j+1}^{\,0,s-1}(x, y)| + |\widehat{f}_{ij}^{\,0s}(x, y)|$$

for all $s \in \{1, \ldots, n\}$, $j \in \{0, \ldots, n-s\}$ and $i \in \{0, \ldots, m\}$, and we also have

$$|\widehat{f}_{i0}^{\,rn}(x, y) - f_{i0}^{rn}(x, y)| \leqslant u\pi_{i0}^{rn}$$

for all $r \in \{1, \ldots, m\}$ and $i \in \{0, 1, \ldots, m-r\}$, where

$$\pi_{i0}^{sn} = (1 - \lambda_i^{(m+1-s)}(x))\pi_{i0}^{s-1,n} + \lambda_i^{(m+1-s)}(x)\pi_{i+1,0}^{s-1,n} + (1 - \lambda_i^{(m+1-s)}(x))|\widehat{f}_{ij}^{\,s-1,n}(x, y)|$$
$$+ \lambda_i^{(m+1-s)}(x)|\widehat{f}_{i+1,j}^{\,s-1,n}(x, y)| + |\widehat{f}_{ij}^{\,sn}(x, y)|,$$

for all $s \in \{1, \ldots, m\}$ and $i \in \{0, 1, \ldots, m-s\}$.

Proof. By step 2 of Algorithm 3, and applying (13) and (14), we have

$$\widehat{f}_{ij}^{\,0s}(x,y)=[(1-\overline{\lambda}_j^{(n+1-s)}(y))\widehat{f}_{ij}^{\,0,s-1}(x,y)(1+\delta_{ij}^{0,s-1})+\overline{\lambda}_j^{(n+1-s)}(y)\widehat{f}_{i,j+1}^{\,0,s-1}(x,y)(1+\delta_{i,j+1}^{0,s-1})]\frac{1}{1+\varepsilon_{ij}^{0s}}$$

for all $i\in\{0,1,\dots,m\}$, $s\in\{1,\dots,n\}$ and $j\in\{0,1,\dots,n-s\}$, where $|\delta_{ij}^{0,s-1}|,|\delta_{i,j+1}^{0,s-1}|,|\varepsilon_{ij}^{0s}|\leqslant u$. So, by the previous formula, taking into account that

$$f_{ij}^{0s}(x,y)=(1-\overline{\lambda}_j^{(n+1-s)}(y))f_{ij}^{0,s-1}(x,y)+\overline{\lambda}_j^{(n+1-s)}(y)f_{i,j+1}^{0,s-1}(x,y)$$

and denoting $R_{ij}^{0s}(x,y):=\widehat{f}_{ij}^{\,0s}(x,y)-f_{ij}^{0s}(x,y)$, we deduce that

$$\begin{aligned}R_{ij}^{0s}(x,y)=&(1-\overline{\lambda}_j^{(n+1-s)}(y))R_{ij}^{0,s-1}(x,y)+(1-\overline{\lambda}_j^{(n+1-s)}(y))\widehat{f}_{ij}^{\,0,s-1}(x,y)\delta_{ij}^{0,s-1}+\overline{\lambda}_j^{(n+1-s)}(y)R_{i,j+1}^{0,s-1}(x,y)\\&+\overline{\lambda}_j^{(n+1-s)}(y)\widehat{f}_{i,j+1}^{\,0,s-1}(x,y)\delta_{i,j+1}^{0,s-1}-\varepsilon_{ij}^{0s}\widehat{f}_{ij}^{\,0s}(x,y).\end{aligned}$$

Then we can write that

$$\begin{aligned}|R_{ij}^{0s}(x,y)|\leqslant&(1-\overline{\lambda}_j^{(n+1-s)}(y))|R_{ij}^{0,s-1}(x,y)|+\overline{\lambda}_j^{(n+1-s)}(y)|R_{i,j+1}^{0,s-1}(x,y)|+u[(1-\overline{\lambda}_j^{(n+1-s)}(y))|\widehat{f}_{ij}^{\,0,s-1}(x,y)|\\&+\overline{\lambda}_j^{(n+1-s)}(y)|\widehat{f}_{i,j+1}^{\,0,s-1}(x,y)|+|\widehat{f}_{ij}^{\,0s}(x,y)|].\end{aligned}\tag{26}$$

Now, taking into account that $R_{ij}^{00}(x,y)=0$ for all $i\in\{0,\dots,m\}$ and $j\in\{0,\dots,n\}$, we will prove by induction on $r\in\{1,\dots,n\}$ that

$$|R_{ij}^{0r}(x,y)|\leqslant u\pi_{ij}^{0r}\quad\text{for all } i\in\{0,1,\dots,m\}\ \text{ and }\ j\in\{0,1,\dots,n-r\},\tag{27}$$

where $\pi_{ij}^{00}=0$ for all $i\in\{0,1,\dots,m\}$ and $j\in\{0,1,\dots,n\}$, and

$$\begin{aligned}\pi_{ij}^{0s}=&(1-\overline{\lambda}_j^{(n+1-s)}(y))\pi_{ij}^{0,s-1}+\overline{\lambda}_j^{(n+1-s)}(y)\pi_{i,j+1}^{0,s-1}+(1-\overline{\lambda}_j^{(n+1-s)}(y))|\widehat{f}_{ij}^{\,0,s-1}(x,y)|\\&+\overline{\lambda}_j^{(n+1-s)}(y)|\widehat{f}_{i,j+1}^{\,0,s-1}(x,y)|+|\widehat{f}_{ij}^{\,0s}(x,y)|\end{aligned}$$

for all $s\in\{1,\dots,n\}$, $j\in\{0,\dots,n-s\}$ and $i\in\{0,\dots,m\}$.

By the formula (26) for $s=1$, taking into account that $R_{ij}^{00}=R_{i,j+1}^{00}=\pi_{ij}^{00}=\pi_{i,j+1}^{00}=0$, we can deduce that

$$\begin{aligned}|R_{ij}^{01}(x,y)|\leqslant&(1-\overline{\lambda}_j^{(n)}(y))|R_{ij}^{00}(x,y)|+\overline{\lambda}_j^{(n)}(y)|R_{i,j+1}^{00}(x,y)|+u[(1-\overline{\lambda}_j^{(n)}(y))|\widehat{f}_{ij}^{\,00}(x,y)|\\&+\overline{\lambda}_j^{(n)}(y)|\widehat{f}_{i,j+1}^{\,00}(x,y)|+|\widehat{f}_{ij}^{\,01}(x,y)|]=u\pi_{ij}^{01},\end{aligned}$$

and so (27) holds for $r=1$. Now let us suppose that (27) holds for $r\in\{1,\dots,n-1\}$ and let us see that it also holds for $r+1$. By (26) for $s=r+1$, we have

$$\begin{aligned}|R_{ij}^{0,r+1}(x,y)|\leqslant&(1-\overline{\lambda}_j^{(n-r)}(y))|R_{ij}^{0r}(x,y)|+\overline{\lambda}_j^{(n-r)}(y)|R_{i,j+1}^{0r}(x,y)|+u[(1-\overline{\lambda}_j^{(n-r)}(y))|\widehat{f}_{ij}^{\,0r}(x,y)|\\&+\overline{\lambda}_j^{(n-r)}(y)|\widehat{f}_{i,j+1}^{\,0r}(x,y)|+|\widehat{f}_{ij}^{\,0,r+1}(x,y)|].\end{aligned}$$

Hence, by the induction hypothesis, we deduce that

$$\begin{aligned}|R_{ij}^{0,r+1}(x,y)|\leqslant&(1-\overline{\lambda}_j^{(n-r)}(y))u\pi_{ij}^{0r}+\overline{\lambda}_j^{(n-r)}(y)u\pi_{i,j+1}^{0r}\\&+u[(1-\overline{\lambda}_j^{(n-r)}(y))|\widehat{f}_{ij}^{\,0r}(x,y)|+\overline{\lambda}_j^{(n-r)}(y)|\widehat{f}_{i,j+1}^{\,0r}(x,y)|\\&+|\widehat{f}_{ij}^{\,0,r+1}(x,y)|]=u\pi_{ij}^{0,r+1},\end{aligned}$$

and the induction holds.

Analogously as we have performed above but using step 1 of Algorithm 3 instead of step 2, substituting indices ${}^{0s}_{ij}$ by indices ${}^{rn}_{i0}$ and parameter x by parameter y, and denoting $R^{rn}_{i0}(x,y) := \widehat{f}^{rn}_{i0}(x,y) - f^{rn}_{i0}(x,y)$, we can prove by induction on $r \in \{1,\dots,m\}$ that

$$|R^{rn}_{i0}(x,y)| \leqslant u\pi^{rn}_{i0} \quad \text{for all } i \in \{0,1,\dots,m-r\}, \tag{28}$$

where

$$\begin{aligned}\pi^{rn}_{i0} = {} & (1-\lambda^{(m+1-r)}_i(x))\pi^{r-1,n}_{i0} + \lambda^{(m+1-r)}_i(x)\pi^{r-1,n}_{i+1,0} + (1-\lambda^{(m+1-r)}_i(x))|\widehat{f}^{r-1,n}_{ij}(x,y)| \\ & + \lambda^{(m+1-r)}_i(x)|\widehat{f}^{r-1,n}_{i+1,j}(x,y)| + |\widehat{f}^{rn}_{ij}(x,y)|.\end{aligned}$$

Let us see that we can reduce the number of operations for the computation of the sequences π^{0s}_{ij} and π^{rn}_{i0} of Theorem 8 by defining the new sequences $M^{00}_{ij} := |\widehat{f}^{00}_{ij}(x,y)|/2$ for all $i \in \{0,1,\dots,m\}$ and $j \in \{0,1,\dots,n\}$,

$$M^{0s}_{ij} := \frac{\pi^{0s}_{ij} + |\widehat{f}^{0s}_{ij}(x,y)|}{2}$$

for all $i \in \{0,1,\dots,m\}$, $s \in \{1,\dots,n\}$ and $j \in \{0,1,\dots,n-s\}$, and

$$M^{rn}_{i0} := \frac{\pi^{rn}_{i0} + |\widehat{f}^{rn}_{i0}(x,y)|}{2}$$

for all $r \in \{1,\dots,m\}$ and $i \in \{0,1,\dots,m-r\}$. As a consequence of Theorem 8 we have

$$M^{0s}_{ij} = (1-\lambda^{(n+1-s)}_j(y))M^{0,s-1}_{ij} + \lambda^{(n+1-s)}_j(y)M^{0,s-1}_{i,j+1} + |\widehat{f}^{0s}_{ij}(x,y)| \tag{29}$$

for all $s \in \{1,\dots,n\}$, $j \in \{0,1,\dots,n-s\}$ and $i \in \{0,1,\dots,m\}$, and we also have

$$M^{rn}_{i0} = (1-\lambda^{(m+1-r)}_i(x))M^{r-1,n}_{i0} + \lambda^{(m+1-r)}_i(x)M^{r-1,n}_{i+1,0} + |\widehat{f}^{rn}_{i0}(x,y)| \tag{30}$$

for all $r \in \{1,\dots,m\}$ and $i \in \{0,1,\dots,m-r\}$.

As a straightforward consequence of formulas (29) and (30) we deduce the following corner cutting algorithm that simultaneously evaluates the surface and obtains an error bound.

Algorithm 9. *Let us consider the following tensor product surface represented in a tensor product system*

$$F(x,y) = \sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij}u_i(x)\overline{u}_j(y).$$

Let us suppose that, in fact, the system is a tensor product corner cutting system. Hence, we can write

$$F(x,y) = \Lambda_1(x)\cdots\Lambda_m(x)(P_{ij})^{0\leqslant j\leqslant n}_{0\leqslant i\leqslant m}\overline{\Lambda}_n(y)^{\mathrm{T}}\cdots\overline{\Lambda}_1(y)^{\mathrm{T}}.$$

Then performing

1. $\widehat{f}^{00}_{ij}(x,y) = P_{ij}$, $M^{00}_{ij}(x,y) = |\widehat{f}^{00}_{ij}(x,y)|/2$ *for* $0 \leqslant i \leqslant m$ *and* $0 \leqslant j \leqslant n$.
2. $\widehat{f}^{0s}_{ij}(x,y) = (1-\overline{\lambda}^{(n+1-s)}_j(y))\widehat{f}^{0,s-1}_{ij}(x,y) + \overline{\lambda}^{(n+1-s)}_j(y)\widehat{f}^{0,s-1}_{i,j+1}(x,y)$,

$$M^{0s}_{ij}(x,y) = (1-\overline{\lambda}^{(n+1-s)}_j(y))M^{0,s-1}_{ij}(x,y) + \overline{\lambda}^{(n+1-s)}_j(y)M^{0,s-1}_{i,j+1}(x,y) + |\widehat{f}^{0s}_{ij}(x,y)|$$

   for $0 \leqslant i \leqslant m$, $1 \leqslant s \leqslant n$ *and* $0 \leqslant j \leqslant n-s$.
3. $\widehat{f}^{rm}_{i0}(x,y) = (1-\lambda^{(m+1-r)}_i(x))\widehat{f}^{r-1,n}_{i0}(x,y) + \lambda^{(m+1-r)}_i(x)\widehat{f}^{r-1,n}_{i+1,0}(x,y)$,

$$M^{rm}_{i0}(x,y) = (1-\lambda^{(m+1-r)}_i(x))M^{r-1,n}_{i0}(x,y) + \lambda^{(m+1-r)}_i(x)M^{r-1,n}_{i+1,0}(x,y) + |\widehat{f}^{rn}_{i0}(x,y)|$$

   for $1 \leqslant r \leqslant m$ *and* $0 \leqslant i \leqslant m-r$.
4. $\mu = u(2M^{mn}_{00} - |\widehat{f}^{mn}_{00}(x,y)|)$ *we have* $\widehat{f}^{mn}_{00}(x,y) = \widehat{F}(x,y)$ *and* $|F(x,y) - \widehat{F}(x,y)| \leqslant \mu$.

Remark 10. We can easily check that the previous algorithm evaluates a scalar function represented through a tensor product corner cutting system and obtains the running error performing $(\frac{1}{2})(3(m+1)n(n+1)+3m(m+1))+1$ sums/subtractions, $2(m+1)n(n+1)+2m(m+1)+2$ products and $(m+1)(n+1)$ quotients. In consequence, the previous algorithm has a computational cost of mn^2 order: the same order as Algorithm 3. Approximately, the number of operations of the previous algorithm is between the double and the triple of the number of operations of Algorithm 3.

5. Numerical experiments and conclusions

We shall compare the three corner cutting evaluation algorithms considered along the paper and the adaptation of the Horner algorithm for the evaluation of tensor product surfaces through numerical experiments. In order to compare the algorithms and see the accuracy of the error bounds at ill conditioned problems we have considered a bivariate polynomial defined on $[0,1]\times[0,1]$, which is a generalization of the univariate polynomial considered by Wilkinson in [17], in the sense that it has all its roots uniformly distributed on $[0,1]\times[0,1]$. So we have a bivariate polynomial that presents stability problems when evaluating at points close to its roots. Then, in the following example we evaluate this bivariate polynomial at 1296 points uniformly distributed on $[0,1]\times[0,1]$ through the three corner cutting algorithms considered along the paper and the Horner-type algorithm for the evaluation of tensor product surfaces, calculating the absolute errors and, in addition, in the case of the corner cutting algorithms, the running errors.

Example 11. Let us consider the bivariate polynomial

$$F(x,y) := \prod_{i=1}^{12}\left(x-\frac{i}{12}\right)\prod_{j=1}^{12}\left(y-\frac{j}{12}\right). \tag{31}$$

Its roots are $\{(x,y)\in\mathbb{R}^2 \mid x=i/12 \text{ or } y=i/12 \text{ for some } i\in\{1,\ldots,12\}\}$. In this example we have evaluated the polynomial $F(x,y)$ through the four algorithms, and we have computed, in the case of the three corner cutting algorithms, the running errors at the points of the mesh $S\times S$ where $S=\{(1/72)+i(1/36)|i=0,1,\ldots,35\}$ using a C compiler with double precision. Let us denote the computed values of $F(x,y)$ by $\widehat{F}(x,y)$. Then, with Mathematica we have computed the exact value of $F(x,y)$ and the absolute errors $|F(x,y)-\widehat{F}(x,y)|$ for each of the four evaluation algorithms.

In Fig. 1 we can see the absolute value of the logarithms of the absolute errors to the base 10 of all the considered evaluation algorithms. In Table 1 we show the absolute error of all the algorithms and the running error (using Algorithm 9) corresponding to the de Casteljau, the Wang–Ball and the DP algorithms for tensor product surfaces at the point of the mesh where the absolute error is maximum for each of the algorithms.

The polynomial considered in Example 11 has serious conditioning problems. As we saw in Fig. 1 and Table 1 of Example 11, although none of the three corner cutting algorithms presents a catastrophic behaviour, the de Casteljau tensor product algorithm has better precision than the other three algorithms. This happens because, by Theorem 7, the Bernstein tensor product basis is better conditioned than the Wang–Ball and the DP tensor product bases. In addition, Fig. 1 also showed that the Horner-type algorithm for tensor product surfaces presents a worse behaviour than the other algorithms. In fact, the maximal absolute error of Horner algorithm is $1.4\cdot 10^{-13}$ (see Table 1). On the other hand, we can also observe in Table 1 that the running errors provide very accurate bounds of the absolute error.

We can observe in Fig. 1 that the Wang–Ball algorithm is more precise than the DP algorithm when evaluating the polynomial $F(x,y)$ in Example 11. But we cannot expect that the Wang–Ball algorithm always behaves better than the DP algorithm for all bivariate polynomials because we cannot establish an order relation between the condition number of these last two bases as in the case of the Wang–Ball and DP tensor product bases with respect to the Bernstein tensor product basis (see Theorem 7). That is due to the fact that matrices D_{mn} and D_{mn}^{-1} such that $a^{mn}=c^{mn}D_{mn}$ and $c^{mn}=a^{mn}D_{mn}^{-1}$ with a^{mn} and c^{mn} defined in Theorem 7 have both positive and negative entries as we can check for low degrees m and n (see [11, Lemma 3.1]). In fact, in the following example we have evaluated another bivariate polynomial through the Wang–Ball, the DP and the Horner-type algorithms for tensor product surfaces and we have computed the corresponding absolute errors in order to illustrate the issues previously mentioned.

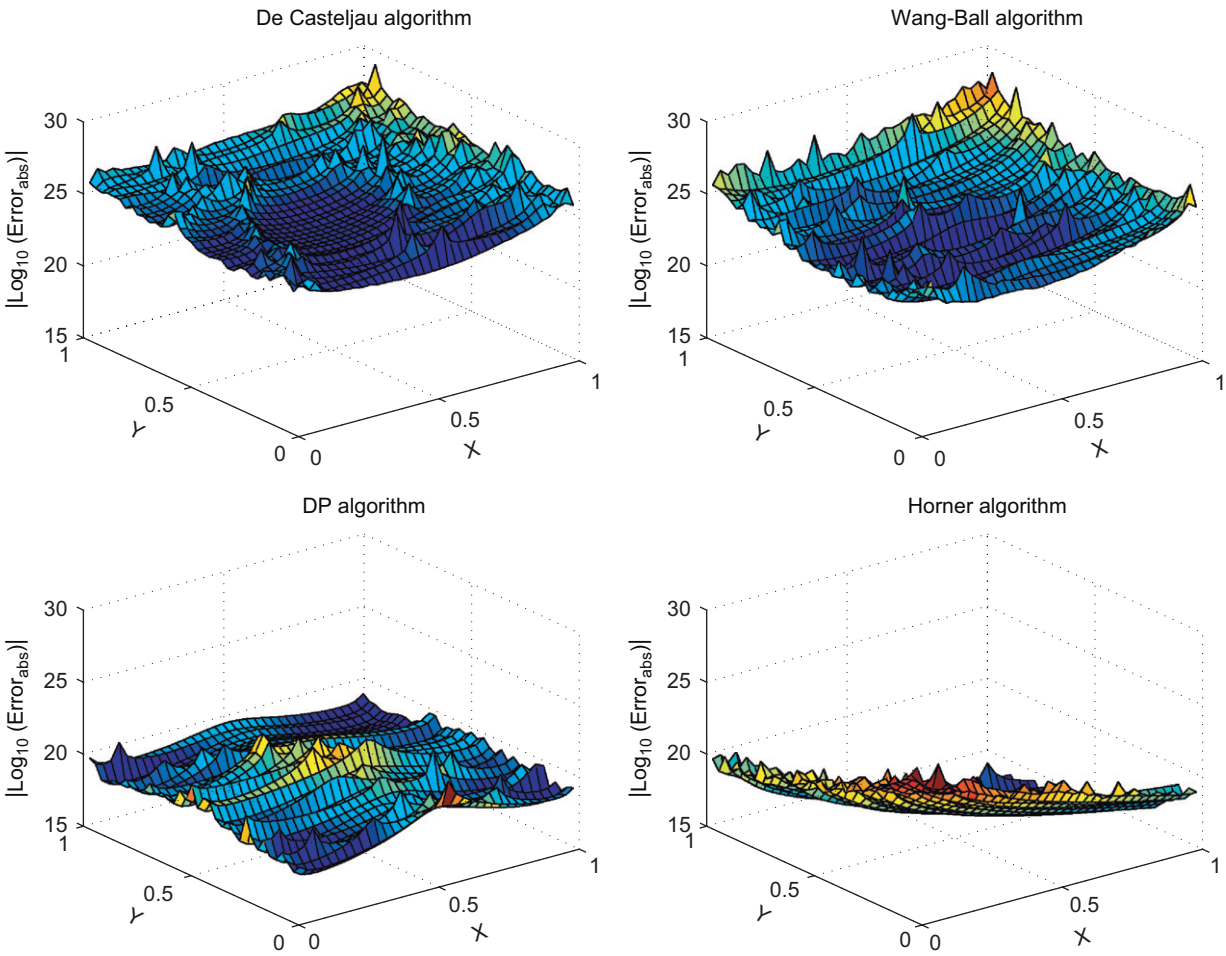

Fig. 1. Absolute value of the logarithms of the absolute errors to the base 10.

Table 1
Maximal absolute and running errors

De Casteljau		Wang–Ball		DP		Horner
Abs. error	Run. error	Abs. error	Run. error	Abs. error	Run. error	Abs. error
$1.4 \cdot 10^{-24}$	$3.3 \cdot 10^{-23}$	$1.3 \cdot 10^{-22}$	$6.3 \cdot 10^{-21}$	$1.2 \cdot 10^{-18}$	$5.2 \cdot 10^{-17}$	$1.4 \cdot 10^{-13}$

Example 12. Let us consider the bivariate polynomial

$$G(x, y) := (x-1)^{12}(y-1)^{12}. \tag{32}$$

Its roots are $\{(x, y) \in \mathbb{R}^2 \mid x=1 \text{ or } y=1\}$. In this example we have evaluated the polynomial $G(x, y)$ at the points of the mesh $T \times T$ where $T = \{i/38 | i = 1, \ldots, 35\}$ using a C compiler with double precision through the DP, the Wang–Ball and the Horner-type algorithms for tensor product surfaces. Let us denote the computed values of $G(x, y)$ by $\widehat{G}(x, y)$. Then, with Mathematica we have computed the exact value of $G(x, y)$ and the absolute errors $|G(x, y) - \widehat{G}(x, y)|$ for each of the evaluation algorithms. In this case, the maximum absolute errors are $1.0 \cdot 10^{-16}$, $2.2 \cdot 10^{-16}$ and $7.9 \cdot 10^{-11}$ for the DP, the Wang–Ball and the Horner-type algorithms for tensor product surfaces, respectively.

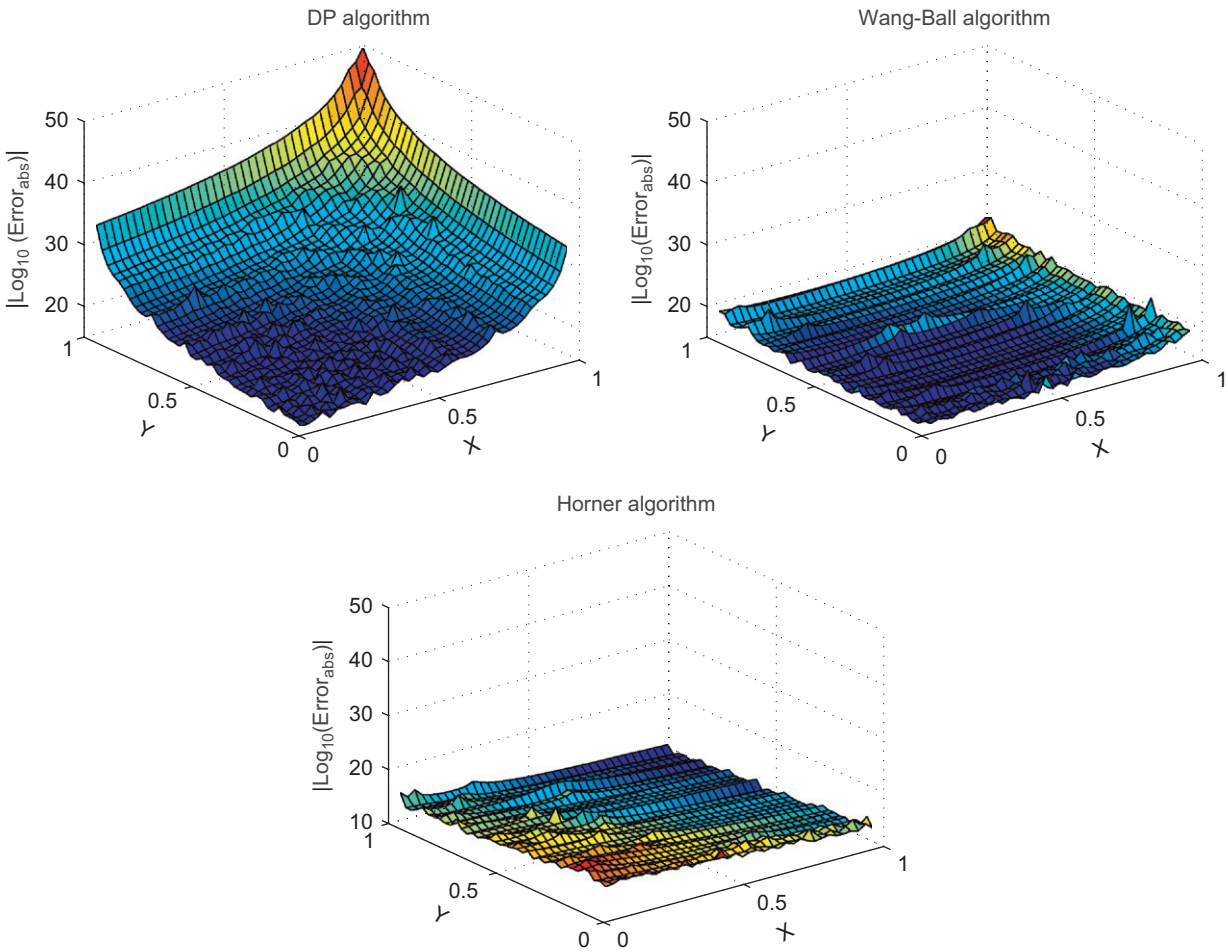

Fig. 2. Absolute value of the logarithms of the absolute errors to the base 10.

In Fig. 2 we can see the absolute value of the logarithms of the absolute errors to the base 10 of the Wang–Ball, the DP and the Horner-type algorithms for tensor product surfaces.

When evaluating the bivariate polynomial $G(x, y)$ in Example 12 the DP algorithm has better precision than the Wang–Ball algorithm and the Horner-type algorithm presents again the worst behaviour of the considered algorithms, as Fig. 2 shows.

Finally we present the conclusions. In this paper, we have considered the corner cutting evaluation algorithms for Wang–Ball and DP tensor product surfaces, which present lower computational cost than the de Casteljau tensor product algorithm. We have proved that the three evaluation algorithms are backward stable. We have also proved that the Bernstein tensor product basis is better conditioned than the Wang–Ball and the DP tensor product bases, although the three algorithms present great accuracy even for pathological polynomials. Since the DP basis presents better shape preserving properties than the Wang–Ball basis (as recalled in the introduction), its use is convenient when we want to control computational cost and shape properties simultaneously. In the numerical examples we have also shown the great accuracy of the running error bound provided by Algorithm 2, which calculates it simultaneously to the evaluation of the surface without increasing significantly the computational cost. Finally, the well-known Horner algorithm presents worse stability properties than the other algorithms considered in the paper.

Acknowledgment

The authors are thankful to the referees for their valuable comments and suggestions.

References

[1] N. Dejdumrong, H.N. Phien, Efficient algorithms for Bezier curves, Comput. Aided Geom. Design 17 (2000) 247–250.
[2] N. Dejdumrong, H.N. Phien, H.L. Tien, K.M. Lay, Rational Wang–Ball curves, Internat. J. Math. Educ. Sci. Technol. 32 (2001) 565–584.
[3] J. Delgado, J.M. Peña, A shape preserving representation with an evaluation algorithm of linear complexity, Comput. Aided Geom. Des. 20 (2003) 1–10.
[4] J. Delgado, J.M. Peña, Corner cutting systems, Comput. Aided Geom. Design 22 (2005) 81–97.
[5] J. Delgado, J.M. Peña, On the generalized Ball bases, Adv. Comput. Math. 24 (2006) 263–280.
[6] G. Farin, Curves and Surfaces for Computer Aided Geometric Design, Academic Press, Inc., San Diego, CA, 1988.
[7] R.A. Horn, C.R. Johnson, Topics in Matrix Analysis, Cambridge University Press, New York, 1995.
[8] J. Hoschek, D. Lasser, Fundamentals of Computer Aided Geometric Design, A.K. Peters, Wellesley, MA, 1993.
[9] S.R. Jiang, G.J. Wang, Conversion and evaluation for two types of parametric surfaces constructed by NTP bases, Math. Comput. Modelling 49 (2005) 321–329.
[10] D. Lasser, Ein neuer Aspekt des de Casteljau Algorithmus, Preprint 853, Fachbereich Mathematik, Technische Hochschule Darmstadt (1984).
[11] T. Lyche, J.M. Peña, Optimally stable multivariate bases, Adv. Comput. Math. 20 (2004) 149–159.
[12] E. Mainar, J.M. Peña, Error analysis of corner cutting algorithms, Numer. Algorithms 22 (1999) 41–52.
[13] J.M. Peña (Ed.), Shape Preserving Representations in Computer Aided Geometric Design, Nova Science Publishers, Commack, NY, 1999
[14] J.M. Peña, On the optimal stability of bases of univariate functions, Numer. Math. 91 (2002) 305–318.
[15] H. Shi-Min, W. Guo-Zhao, J. Tong-Guang, Properties of two types of generalized Ball curves, Comput. Aided Des. 28 (1996) 125–133.
[16] G.J. Wang, Ball curve of high degree and its geometric properties, Appl. Math.: A Journal of Chinese Universities 2 (1987) 126–140.
[17] J.H. Wilkinson, The evaluation of the zeros of ill-conditioned polynomials Parts I and II, Numer. Math. 1 (1959) 150–166 & 167–180.