Accurate evaluation of Bézier curves and surfaces and the Bernstein-Fourier algorithm

Jorge Delgado [b] J. M. Peña [a]

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

[b] *Departamento de Matemática Aplicada, Universidad de Zaragoza, Spain*
Email address: `jorgedel@unizar.es`*; Phone number: +34978618174*

Abstract

The Bernstein-Fourier algorithm for the evaluation of polynomial curves is extended for the evaluation of polynomial tensor product surfaces. Under a natural hypothesis, accurate evaluation of Bézier curves and surfaces through several algorithms is discussed. Numerical experiments comparing the accuracy of the corresponding Horner, de Casteljau, VS and Bernstein-Fourier algorithms are presented.

Key words: Bézier curves and surfaces; Bernstein-Fourier algorithm; polynomial evaluation

1 Introduction

In [2] L. H. Bezerra introduced a new method to evaluate Bézier polynomials, that is, polynomials of the form

$$p(t) = \sum_{i=0}^{k} c_i b_i^k(t), \quad t \in [0,1], \tag{1}$$

where the polynomials $b_i^k(s) = \binom{k}{i} s^i (1-s)^{k-i}$ are known as *Bernstein basis polynomials* of degree k ($k \geq 0$). The new method is based on using a so-called Bernstein-Fourier representation. So, from now on, we will call Bernstein-Fourier algorithm to the new algorithm. The author also carried out numerical experiments in order to check the performance of the new method with respect to computational cost and accuracy. In addition, a comparison of these two aspects, in particular the computational cost, of the new method with respect to the usual de Casteljau and VS algorithms for the evaluation of Bézier curves was also performed. Assuming positive control points, the numerical experiments with the three algorithms were accurate.

In [4] and [7] some of the most usual algorithms for the evaluation of polynomial curves were compared from the point of view of accuracy. In this paper we complete the numerical experiments in [2], [4] and [7] comparing the de Casteljau algorithm, the VS, the Horner and the Bernstein-Fourier algorithms. We also explain the theoretical reasons for expecting accuracy for the numerical experiments of [2] when using VS and de Casteljau algorithms. In addition, we will carry out the generalization of the Bernstein-Fourier algorithm for the evaluation of Bézier tensor product surfaces and then we will compare it numerically with the corresponding versions for tensor products of the de Casteljau, the VS and the Horner algorithms.

In Section 2, de Casteljau and VS algorithms are recalled. It is also justified why many algorithms in Computer Aided Design evaluate polynomial curves to high relative accuracy under the natural hypothesis (in this field) of starting with positive control points. In Section 3, we consider ill-conditioned polynomials to compare univariate Horner, de Casteljau, VS and Bernstein-Fourier algorithms. In Section 4, evaluation algorithms for polynomial tensor product surfaces are derived and their high relative accuracy is discussed. We particularize the cases of Bernstein-Fourier, de Casteljau and VS algorithms. Section 5 presents numerical examples in order to compare the algorithms.

2 Accuracy in C.A.G.D.

In [2] L. H. Bezerra performed the numerical experiments assuming that all control points are positive. This assumption avoided ill-conditioned polynomials. In this section, we shall show that this is a natural assumption in Computer Aided Geometric Design (from now on, C.A.G.D.) and that it permits to assure high relative precision for the evaluation through a large family of representations in C.A.G.D.

We say that we know a quantity with *high relative accuracy* if its relative error is bounded by $\mathcal{O}(\varepsilon)$, where ε is the machine precision. Given an algebraic expression defined by additions, subtractions, multiplications and divisions and assuming that each initial real datum is known to high relative accuracy, then it is well known that the algebraic expression can be computed accurately if it is defined by sums of numbers of the same sign, products and quotients (cf. p. 52 of [8]). In other words, the only forbidden operation is true subtraction, due to possible cancellation in leading digits. Moreover, in a (well-implemented) floating point arithmetic high relative accuracy is also preserved even when we perform true subtractions when the operands are original (and so, exact) data (cf. p. 53 of [8]).

Let us recall that the VS algorithm evaluates polynomials of degree n represented in the form

$$p(t) = \sum_{i=0}^{n} c_i t^i (1-t)^{n-i}, \quad t \in [0,1], \tag{2}$$

with a computational cost of $2n-1$ products, $2n-1$ additions and 1 quotient. This algorithm has a nested nature like Horner and was introduced for bivariate triangular polynomials in [10]. In fact, taking into account that

$$p(t) = \sum_{i=0}^{n} c_i t^i (1-t)^{n-i} = t^n \sum_{i=0}^{n} c_i \left(\frac{1-t}{t}\right)^{n-i} = (1-t)^n \sum_{i=0}^{n} c_i \left(\frac{t}{1-t}\right)^{n-i},$$

VS algorithm computes $p(t)$ applying Horner algorithm at points $\frac{1-t}{t}$ (if $t \geq 1/2$) or $\frac{t}{1-t}$ (if $t < 1/2$), and then multiplying the obtained result by t^n or $(1-t)^n$, respectively. For more details in the algorithm see for example [7]. Observe that Horner algorithm and VS algorithm can be computed accurately when the control points are positive. The same conclusion holds for the de Casteljau algorithm and, more generally, for corner cutting algorithms as we shall show in the following paragraphs.

In C.A.G.D., given a system of functions $(u_0, \ldots, u_n)$ defined on a compact interval $I = [a,b]$, and points $P_0, \ldots, P_n \in \mathbf{R}^s$, a parametric curve γ is defined by:

$$\gamma(t) = \sum_{i=0}^{n} u_i(t) P_i. \tag{3}$$

The points are called *control points* of γ, and the polygon $P_0 \cdots P_n$ is called *control polygon* of γ. We now recall an important family of systems in C.A.G.D. called corner cutting systems in [5]. Let us denote by $\Lambda_i(t)$ the bidiagonal matrix

$$\begin{pmatrix} 1-\lambda_0^{(i)}(t) & \lambda_0^{(i)}(t) & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1-\lambda_{i-1}^{(i)}(t) & \lambda_{i-1}^{(i)}(t) \end{pmatrix} \tag{4}$$

for all $i \in \{1, \ldots, n\}$, where $\lambda_j^{(i)} : [a,b] \to [0,1]$ for $0 \leq j < i \leq n$. We say that $\Lambda_1(t) \cdots \Lambda_n(t)$ is a corner cutting representation on $[a,b]$ of the system of functions $(u_0, \ldots, u_n)$ defined on $[a,b]$ if

$$(u_0(t), \ldots, u_n(t)) = \Lambda_1(t) \cdots \Lambda_n(t), \tag{5}$$

where the matrices $\Lambda_i(t)$ are given by (4) for all $i \in \{1, \ldots, n\}$ and the functions $\lambda_j^{(i)} : [a,b] \to [0,1]$ are continuous and increasing for all $0 \leq j < i \leq n$ (and are called *cutting functions*). A system of functions admitting a corner cutting representation on $[a,b]$ will be called a *corner cutting system* on $[a,b]$.

By (5), any function

$$f(t) = \sum_{i=0}^{n} c_i u_i(t) \tag{6}$$

can be evaluated by the corner cutting algorithm provided by

$$\Lambda_1(t) \cdots \Lambda_n(t)(c_0, \dots, c_n)^T = f(t). \tag{7}$$

In fact, the evaluation of the curve given in (3) can be reduced, componentwise, to s evaluations as in (6). Let us also denote by $(\mathbf{R}^s)_+$ the set of points of $\mathbf{R}^s$ such that all their components are positive (for instance, if $s = 3$, then $(\mathbf{R}^s)_+$ is the positive octant). Observe that, for design purposes, in order to draw any curve, we can take all control points belonging to $(\mathbf{R}^s)_+$. So, if we can calculate accurately $\lambda_j^{(i)}(t)$ and $1 - \lambda_j^{(i)}(t)$ for all i, j, then we also can perform accurately the evaluation of each component of the parametric curve γ (3) through the corner cutting algorithm given by (7) because we are always multiplying nonnegative numbers and summing nonnegative numbers.

Due to the arguments of the previous paragraphs, we deduce that the following evaluation algorithms of polynomial curves (of degree n) can be performed with high relative accuracy when all control points belong to $(\mathbf{R}^s)_+$. Observe that all satisfy that the cutting functions $\lambda_j^{(i)}(t)$ are either a nonnegative constant or the given parameter t (and so each $1 - \lambda_j^{(i)}(t)$ can also be computed accurately because it is a subtraction of two exact data):

- The de Casteljau algorithm, which uses $\lambda_j^{(i)}(t) = t$ for all i, j. It is associated to the Bernstein basis and is the most usual algorithm for the evaluation of Bézier curves. It evaluates a polynomial Bézier curve of degree k with a computational cost of $\mathcal{O}(k^2)$ elementary operations. The best well known evaluation algorithm is the Horner algorithm, which evaluates a k degree polynomial with a computational cost of $\mathcal{O}(k)$ elementary operations in contrast to the $\mathcal{O}(k^2)$ computational cost of de Casteljau. So in the last years two more efficient alternatives to the de Casteljau algorithm have been researched in the literature. These alternatives are VS algorithm and Bernstein-Fourier algorithm.
- The evaluation algorithm of the Wang-Ball basis, which uses (cf. [11])

$$\lambda_j^{(i)}(t) = 0, \quad \text{for } j \in \left\{0, \dots, \left[\frac{i-3}{2}\right]\right\}, \tag{8}$$

$$\lambda_j^{(i)}(t) = t\ , \quad \text{for } j \in \left\{\left[\frac{i-1}{2}\right], \left[\frac{i}{2}\right]\right\}, \tag{9}$$

$$\lambda_j^{(i)}(t) = 1, \quad \text{for } j \in \left\{\left[\frac{i}{2}\right] + 1, \dots, i-1\right\}, \tag{10}$$

  where $[r]$ denotes the greatest positive integer less than or equal to r.
- The evaluation algorithm of the basis introduced in [3] and called DP basis (cf. [9]), which uses

$$\lambda_0^{(i)}(t) = \lambda_{i-1}^{(i)}(t) = t \ , \tag{11}$$

$$\lambda_j^{(i)}(t) = 1, \quad j = 1, \dots, \left[\frac{i}{2}\right] - 1, \tag{12}$$

$$\lambda_j^{(i)}(t) = 0, \quad j = \left[\frac{i+1}{2}\right], \dots, i-2, \tag{13}$$

and, in addition, if $i \geq 3$ is odd, $\lambda_{\frac{i-1}{2}}^{(i)}(t) = \frac{1}{2}$.

- The evaluation algorithm of the pruned curves (see [1]), which also use $\lambda_j^{(i)}(t) = t$ or $\lambda_j^{(i)}(t) = 1$ or $\lambda_j^{(i)}(t) = 0$.

The extension of the evaluation algorithms for Bézier curves to the evaluation algorithms considered in Section 4 for Bézier surfaces inherits the accuracy under positive control points. Therefore, from now on, we shall focus on numerical examples with ill-condioned polynomials and whose coefficients are not necessarily positive, in the univariate as well as in the bivariate case.

3 Numerical examples with ill-conditioned univariate polynomials

First we shall compare the Bernstein-Fourier algorithm of [2] (see the tensor product version in Section 4) for the evaluation of polynomial curves with the de Casteljau, the VS and the Horner algorithms. In [12] and [13] Wilkinson considered the following two ill-conditioned polynomials:

$$f(t) = \prod_{i=1}^{20}\left(t - \frac{i}{20}\right) \quad \text{and} \quad g(t) = \prod_{i=1}^{20}\left(t - \frac{2}{2^i}\right).$$

We have computed, using Mathematica with infinite precision, the exact coefficients of both polynomials with respect to the Bernstein basis and the monomial basis. Then we have evaluated both polynomials $f(t)$ and $g(t)$ by using the four algorithms with the standard IEEE 754 double precision arithmetic at the points of the sequences given by $\mathcal{U} := \{\frac{1}{72} + \frac{i}{36}\}_{i=0}^{35}$ and $\mathcal{V} = \{\frac{i}{39}\}_{i=1}^{38}$, respectively. Finally, we have obtained the exact values of both polynomials at the points of the two meshes by symbolic computation, and the corresponding relative and absolute errors for the four algorithms. The relative errors when evaluating f and g are shown in Figure 1 (a) and (b), respectively. On the other hand, the absoute errors are shown in Figure 2 (a) and (b). As we can see the absolute errors for the four algorithms are low but, since the values of the Wilkinson polynomials in $[0, 1]$ are small numbers, not all algorithms yield low relative errors. In particular, we can observe that the Horner algorithm has less accuracy than the other algorithms for the polynomial $f(t)$ and that the Bernstein-Fourier algorithm has less accuracy than the other algorithms for the polynomial $g(t)$.

Finally, for comparing the four algorithms for the evaluation of polynomials we have considered the following ill-conditioned polynomial:

$$h(t) = \left(t - \frac{1}{2}\right)^{20}.$$

We have repeated the same process than for polynomials f and g, but now at the points of the mesh $\left\{\frac{4i}{100}\right\}_{i=1}^{24}$. We have obtained the relative errors and the absolute errors shown in Figures 3 and 4, respectively. We can observe that, in this case, the algorithm with a good behaviour everywhere is the de Casteljau algorithm.

4 Evaluation of tensor product surfaces

Let us consider a sequence $(P_{ij})_{0\le i\le m}^{0\le j\le n}$ of points in the space. Then a *tensor product Bézier surface* is defined as

$$F(x,y) = \sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij} b_i^m(x) b_j^n(y). \tag{14}$$

Since

$$F(x,y) = \sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij} b_i^m(x) b_j^n(y) = \sum_{i=0}^{m}(\sum_{j=0}^{n} P_{ij} b_j^n(y)) b_i^m(x)$$

and denoting $f_i(y) := \sum_{j=0}^{n} P_{ij} b_j^n(y)$, for $i = 0, 1, \dots, m$, we have

$$F(x,y) = \sum_{i=0}^{m} f_i(y) b_i^m(x). \tag{15}$$

Hence, taking into account the previous reasoning we can evaluate the tensor product surface (14) at a point $(x, y) \in [0,1] \times [0,1]$ by evaluating the $m+1$ Bézier polynomial curves $f_i(y)$, $i = 0, 1, \dots, m$, of degree n at point y, and then evaluating the polynomial Bézier curve of m degree represented in the Bernstein basis polynomial of the same degree with control points $f_0(y), f_1(y), \dots, f_m(y)$ (15) at point x. Observe that, if the control points P_{ij} are positive, then the high relative accuracy of the algorithm is assured when the corresponding curve evaluation algorithms satisfy the same property. Using the previous approach we can extend the univariate algorithms to the tensor product case.

The *Bernstein matrix* of order $k+1$, $B_k(s)$, is the lower triangular matrix whose (i,j) entry is given by $b_{j-1}^{i-1}(s)$ for each $k \geq i \geq j \geq 1$, that is,

$$B_k(s) = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1-s & \binom{1}{1}s & 0 & \cdots & 0 \\ (1-s)^2 & \binom{2}{1}(1-s)s & \binom{2}{2}s^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ (1-s)^k & \binom{k}{1}(1-s)^{k-1}s & \binom{k}{2}(1-s)^{k-2}s^2 & \cdots & \binom{k}{k}s^k \end{pmatrix}$$

Let us consider the Bézier polynomial of degree k given by $p(s) = \sum_{i=0}^{k} c_i\, b_i^k(s)$ with $c_i \in \mathbf{R}$ for all $i \in \{0, 1, \ldots, n\}$. The previous polynomial can be expressed as

$$p(s) = e_{k+1}^T B_k(s)\mathbf{c}, \tag{16}$$

where $e_{k+1} = (0, \ldots, 0, 1)^T \in \mathbf{R}^{k+1}$ and $\mathbf{c} = (c_0, c_1, \ldots, c_k)^T$. Given $w_k = e^{-\frac{2\pi i}{k+1}}$ we consider the $(k+1) \times (k+1)$ Fourier matrix

$$W_k = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_k & (w_k)^2 & \cdots & (w_k)^k \\ 1 & (w_k)^2 & (w_k)^4 & \cdots & (w_k)^{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (w_k)^k & (w_k)^{2k} & \cdots & (w_k)^{k^2} \end{pmatrix},$$

which is a particular case of Vandermonde matrix. From (16), since W_k is an invertible matrix, we have

$$p(s) = e_{k+1}^T B_k(s) W_k((W_k)^{-1}\mathbf{c}).$$

Denoting $\mathbf{u} := (W_k)^{-1}\mathbf{c} = (u_0, u_1, \ldots, u_k)$ and taking into account that

$$e_{k+1}^T B_k(s) W_k = ((1-s+s)^k, (1-s+w_k s)^k, \ldots, (1-s+(w_k)^k s)^k)$$

we have

$$p(s) = \sum_{i=0}^{k} u_i (1 + s((w_k)^i - 1))^k.$$

According to the previous formula and taking into account that $\mathbf{u}$ can be computed applying the inverse fast Fourier transform to $\mathbf{c}$ and the previous discussion in this section about the evaluation of a Bézier tensor product surface $F(x,y)$, we provide the following pseudo-code for its evaluation:

Algorithm 1 *BerFourEVAL algorithm* for the evaluation of a polynomial by the Bernstein-Fourier algorithm

```
Require: $(P_{ij})_{0 \le i \le m}^{0 \le j \le n}$, $(x, y) \in [0,1] \times [0,1]$
Ensure: $F(x,y) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} b_i^m(x) b_j^n(y)$
  for $i = 0$ to $m$ do
    $\mathbf{P}_i = (P_{i0}, P_{i1}, \ldots, P_{in})^T$
    $\mathbf{U}_i = ifft(\mathbf{P}_i)$
    $f_i(y) = \sum_{j=0}^{n} (\mathbf{U}_i)_j (1 + y((w_n)^j - 1))^n$
  end for
  $\mathbf{f} = (f_0(y), f_1(y), \ldots, f_m(y))$
  $\mathbf{V} = ifft(\mathbf{f})$
  $F(x,y) = \sum_{i=0}^{m} \mathbf{V}_i (1 + x((w_m)^i - 1))^m$
```

The previous algorithm requires a computational cost of $\mathcal{O}(n \log n)$ elementary operations (see [2] for more details in the univariate case).

4.2 The de Casteljau algorithm

The de Casteljau algorithm evaluates polynomials of degree n represented in the form (1), with a computational cost of $n(n-1)$ sums and $2n(n-1)$ products. Taking into account the discussion at the beginning of the current section we can obtain a de Casteljau algorithm for the evaluation of the corresponding tensor product functions (see also [6]). So, the computational cost of the corresponding tensor product algorithm is of $(m+1)n(n-1)+m(m-1)$ sums and $2(m+1)n(n-1)+2m(m-1)$ products.

4.3 The VS algorithm

The VS algorithm evaluates polynomials of degree n represented in the form (2). Taking into account the discussion at the beginning of the current section, we can obtain a VS algorithm for the evaluation of the corresponding tensor product polynomials. From the computational cost of the univariate VS algorithm seen in Section 2, we can conclude that the computational cost for the corresponding tensor product VS algorithm is of $(2n-1)(m+1)+(2m-1)$ sums, $(2n-1)(m+1)+(2m-1)$ products and $m+2$ quotients.

5 Bivariate numerical experiments

Now we shall compare the adaptation of the de Casteljau, the VS, the Horner and the Bernstein-Fourier algorithms for the evaluation of tensor product surfaces, which have been implemented according to the approach discussed in Section 4. In order to compare the algorithms and see the accuracy at ill conditioned problems we have considered two bivariate polynomials defined on $[0,1] \times [0,1]$, which are generalizations of the two univariate polynomials considered by Wilkinson in [12] and [13], in the sense that the first tensor product polynomial has all its roots uniformly distributed on $[0,1] \times [0,1]$ and the second one has the most of its roots localized very close to the point $(0,0)$. The particular functions are the following:

$$F(x,y) = \prod_{i=1}^{12}\left(x - \frac{i}{12}\right)\prod_{j=1}^{12}\left(y - \frac{j}{12}\right) \text{ and } G(x,y) = \prod_{i=1}^{12}\left(x - \frac{2}{2^i}\right)\prod_{j=1}^{12}\left(y - \frac{2}{2^j}\right)$$

So we have two tensor product polynomials that present stability problems when evaluating at points close to its roots. First we have evaluated the function $F(x,y)$ at 1296 points uniformly distributed on $[0,1] \times [0,1]$ with the four algorithms considered in IEEE754 double precision. Figure 5 shows the corresponding relative errors and Figure 6 the corresponding absolute errors, where the exact values for the polynomials have been obtained by symbolic computation.

We can observe that the four algorithms behave in a very similar way respect to accuracy than its univarite versions with $f(t)$.

Then we have evaluated the function $G(x,y)$ at 1444 points uniformly distributed on $[0,1] \times [0,1]$ with the four algorithms considered in IEEE754 double precision. Figure 7 shows the corresponding relative errors and Figure 8 shows the corresponding absolute errors. We can observe that the four algorithms behave in a very similar way respect to accuracy than its univarite versions with $g(t)$.

In order to see in detail the difference in the relative errors when evaluating $G(x,y)$ we have computed the mean of the relative errors at the mesh of points for each of the algortihms. For the de Casteljau algorithm the average relative error is $3.37e-13$, for the VS algorithm is $1.61e-13$, for the Horner algorithm $7.87e+01$ and for the Bernstein-Fourier $1.03e-2$.

Finally, we have considered the tensor product polynomial

$$H(x,y) = (x - 1/2)^{12}(y - 1/2)^{12},$$

which is a very ill-conditioned polynomial because of its roots. Then we have evaluated the function $H(x,y)$ at 1444 points uniformly distributed on $[0,1]\times[0,1]$ with the four algorithms considered in IEEE754 double precision. Figure 9 shows the corresponding relative errors and Figure 10 shows the corresponding absolute errors. We can observe that in this case only de Casteljau algorithm has a satisfactory behaviour.

In order to see in detail the difference in the relative errors when evaluating $H(x,y)$ we have computed the mean of the relative errors at the mesh of points for each of the algortihms. For the de Casteljau algorithm the average relative error is $5.48e-15$, for the VS algorithm is $2.09e+18$, for the Horner algorithm $2.25e+23$ and for the Bernstein-Fourier $1.07e+04$.

Acknowledgements

This work has been partially supported by the Spanish Research Grant MTM2012-31544 and by Gobierno de Aragón and Fondo Social Europeo.

References

[1] P.J. Barry, T.D. DeRose and R.N. Goldman, Pruned Bezier Curves, Proceedings Graphics Interface 90 (1990) 229-238.

[2] L.H. Bezerra, Efficient computation of Bézier curves from their Benstein-Fourier representation, Appl. Math. Comput. 220 (2013) 235-238.

[3] J. Delgado and J.M. Peña, A shape preserving representation with an evaluation algorithm of linear complexity, Comput. Aided Geom. Design 20 (2003) 1-10.

[4] J. Delgado and J.M. Peña, On efficient algorithms for polynomial evaluation in CAGD, Monogr. Semin. Mat. García de Galdeano 31 (2004) 111-120.

[5] J. Delgado and J.M. Peña, Corner cutting systems, Comput. Aided Geom. Design 22 (2005) 81-97.

[6] J. Delgado and J.M. Peña, Error analysis of efficient evaluation algorithms for tensor product surfaces, J. Comput. Appl. Math. 219 (2008) 156-169.

[7] J. Delgado and J.M. Peña, Running Relative Error for the Evaluation of Polynomials, SIAM J. on Sci. Computing 31 (2009) 3905-3921.

[8] J. Demmel, M. Gu, S. Eisenstat, I. Slapnicar, K. Veselic, and Z. Drmac, Computing the singular value decomposition with high relative accuracy, Linear Algebra Appl. 299 (1999) 21-80.

[9] C. Jie and G.-J. Wang, Construction of triangular DP surface and its application, J. Comput. Appl. Math. 219 (2008) 312-326.

[10] L. L. Schumaker and W. Volk, Efficient evaluation of multivariate polynomials, Computer Aided Geometric Design 3 (1986) 149-154.

[11] H. Shi-Min, W. Guo-Zhao and Tong-Guang, Properties of two types of generalized Ball curves, Computer-Aided Design 28 (1996) 125-133.

[12] J. H. Wilkinson, The evaluation of the zeros of ill-conditioned polynomials, Parts I and II, Numer. Math. 1 (1959) 150-166 and 167-180.

[13] J. H. Wilkinson, Rounding Errors in Algebraic Processes, Notes on Applied Science, Vol. 32. Her Majesty's Stationery Office, London, 1963.

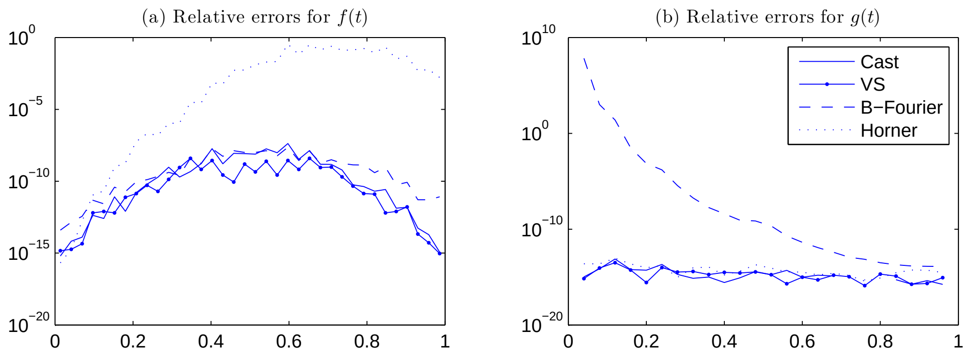

Fig. 1. Relative errors for Wilkinson polynomials

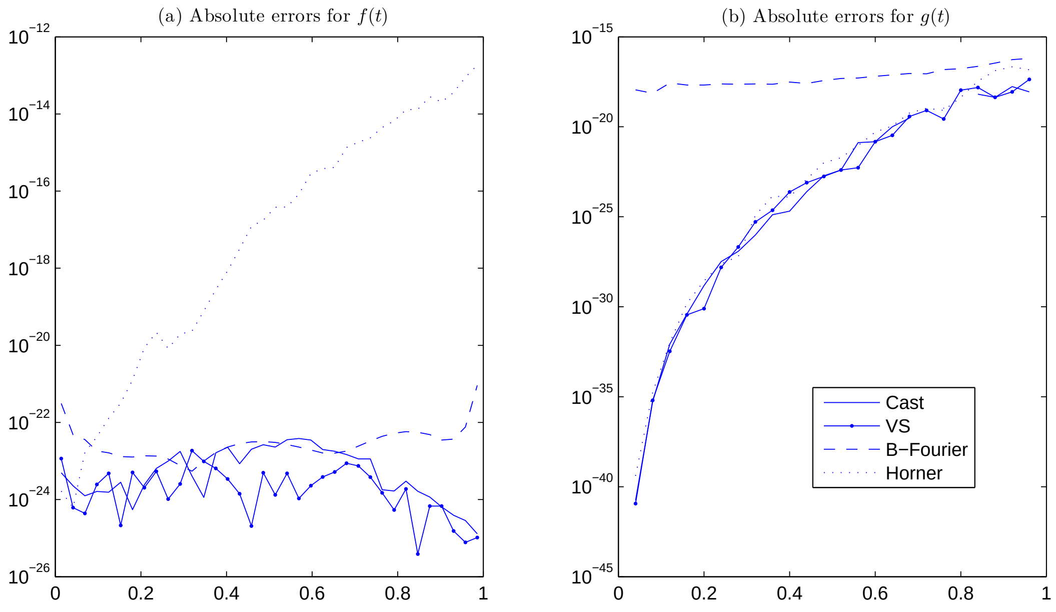

Fig. 2. Absolute errors for Wilkinson polynomials

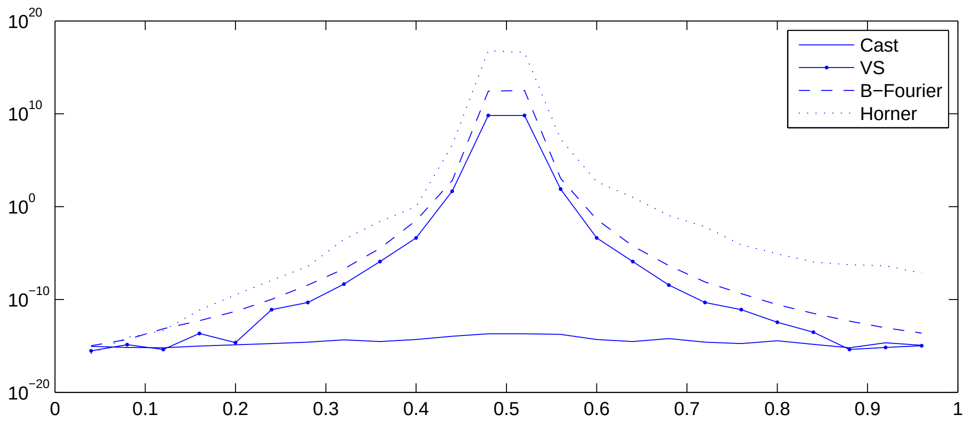

Fig. 3. Relative errors for $h(t)$

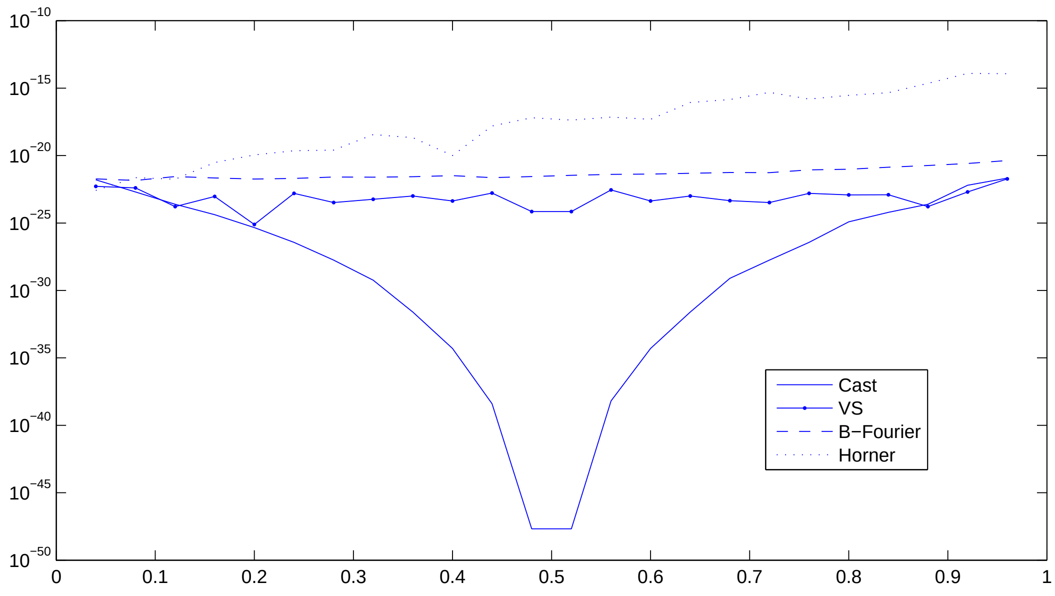

Fig. 4. Absolute errors for $h(t)$

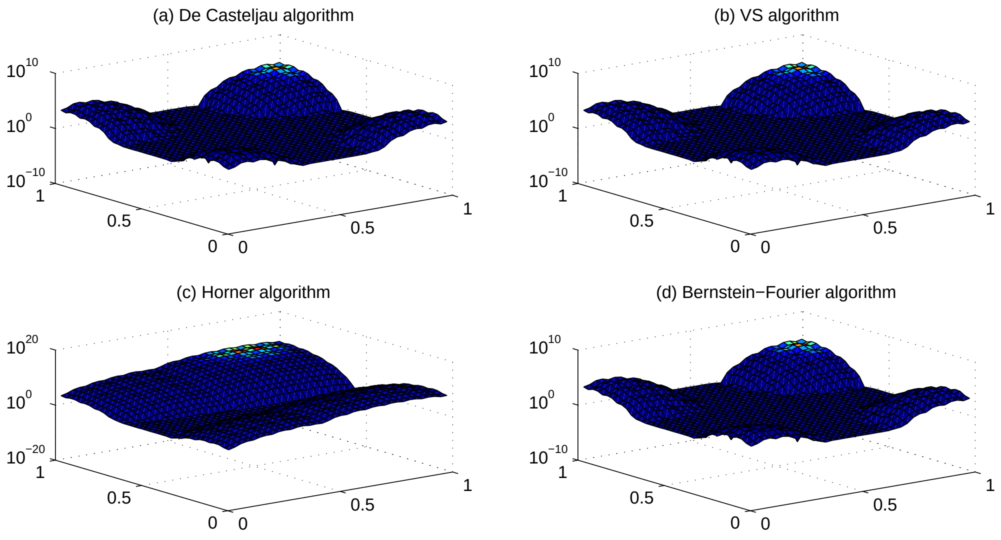

Fig. 5. Relative errors for $F(x, y)$

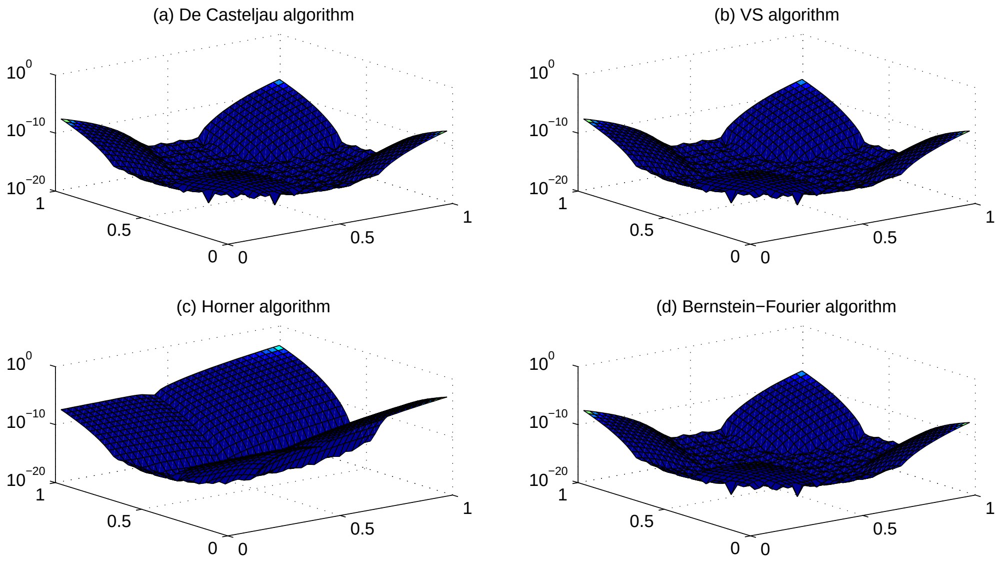

Fig. 6. Absolute errors for $F(x, y)$

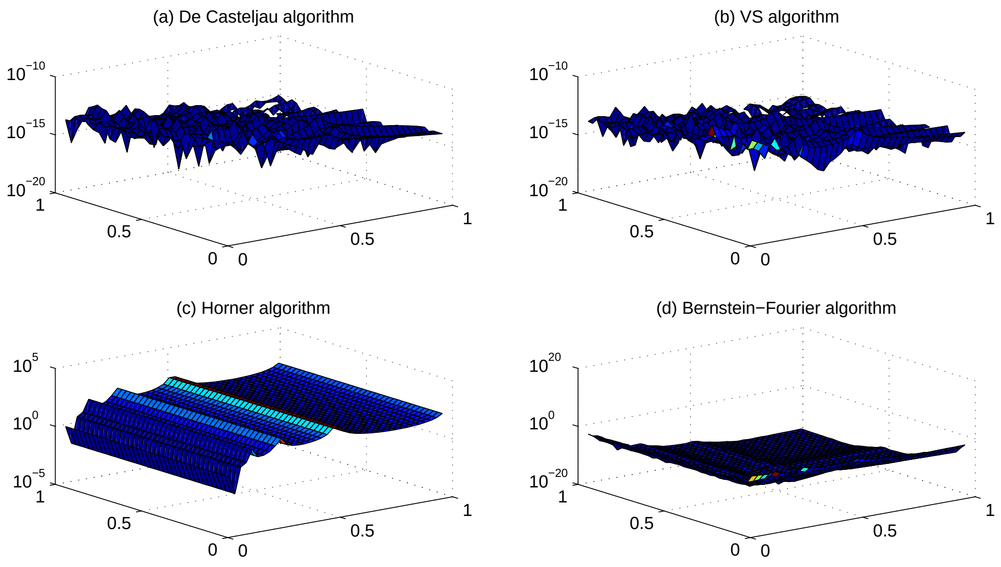

Fig. 7. Relative errors for $G(x, y)$

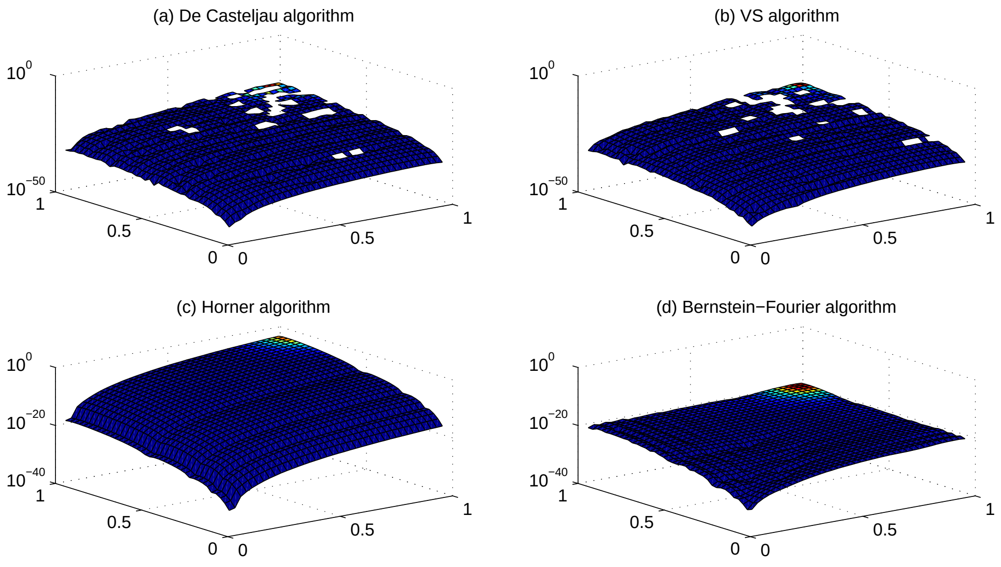

Fig. 8. Absolute errors for $G(x, y)$

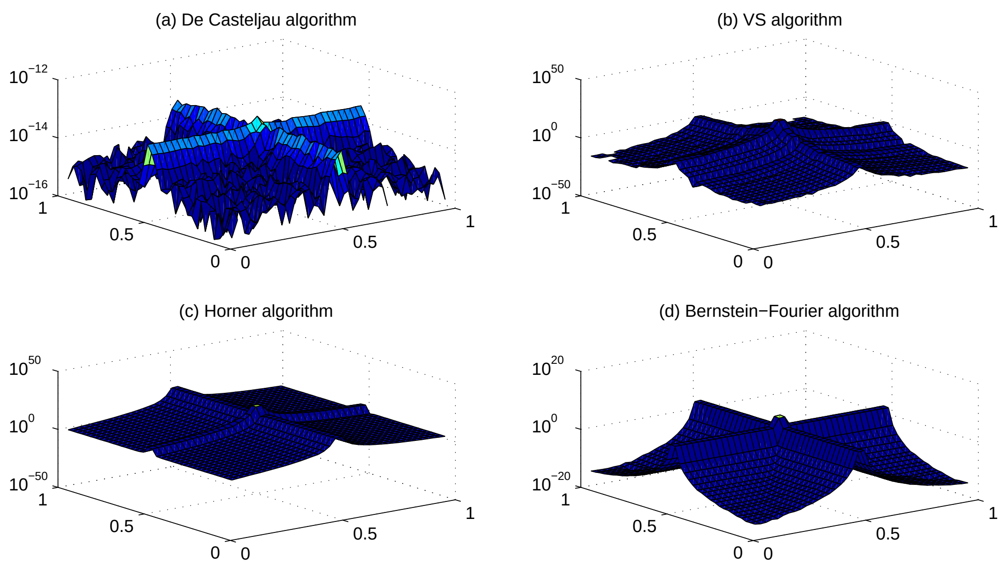

Fig. 9. Relative errors for $H(x,y)$

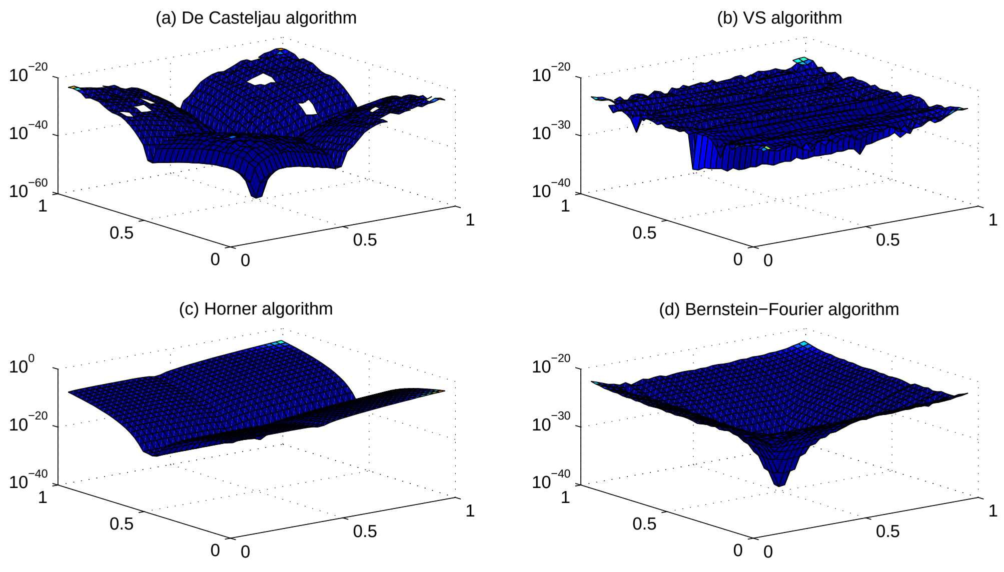

Fig. 10. Absolute errors for $H(x,y)$