On computing Bézier curves by Pascal matrix methods

Licio Hernanes Bezerra [a,*] Leonardo Koller Sacht [b]

[a] *Universidade Federal de Santa Catarina, Departamento de Matemática, Florianópolis, SC 88040-900, Brazil*

[b] *Instituto de Matemática Pura e Aplicada, Rio de Janeiro, RJ 22460-320, Brazil*

Abstract

The main goal of the paper is to introduce methods that compute Bézier curves faster than Casteljau's method does. These methods are based on the spectral factorization of an $n \times n$ Bernstein matrix, $B_n^e(s) = P_n G_n(s) P_n^{-1}$, where P_n is the $n \times n$ lower triangular Pascal matrix. To that end, we first calculate the exact optimum positive value t in order to transform P_n into a scaled Toeplitz matrix (how to do so is a problem that was partially solved by X. Wang and J. Zhou (2006)). Then, fast Pascal matrix-vector multiplications are combined with polynomial evaluations to compute Bézier curves. Nevertheless, when n increases, we need more precise Pascal matrix-vector multiplications to achieve stability in the numerical results. We see here that a Pascal matrix-vector product, combined with a polynomial evaluation and some affine transforms of the vectors of coordinates of the control points, will yield a method that can be used to efficiently compute a Bézier curve of degree n, $n \leq 60$.

Key words: Pascal matrix, Bernstein polynomial, Bézier curve, Toeplitz matrix
1991 MSC: 15A18, 65F15, 68U07

* Corresponding author.
Email addresses: licio@mtm.ufsc.br (Licio Hernanes Bezerra), leo-ks@impa.br (Leonardo Koller Sacht).

1 Introduction

Bézier has his name on the curve B of degree $n-1$ defined from n given points $Z_0 = (x_0, y_0)$, $Z_1 = (x_1, y_1)$, ..., $Z_{n-1} = (x_{n-1}, y_{n-1})$ in $\mathbb{R}^2$ as follows:

$$B(s) = (x(s), y(s)) = \sum_{i=0}^{n-1} Z_i b_{i,n-1}(s), \qquad s \in [0,1],$$

where $b_{i,n-1}(s) = \binom{n-1}{i} s^i (1-s)^{n-1-i}$ for each $i \in \{0, ..., n-1\}$. These are the so-called Bernstein polynomials [2]. Hence, for each $s \in [0,1]$, we have

$$x(s) = \sum_{i=0}^{n-1} b_{i,n-1}(s) x_i, \quad y(s) = \sum_{i=0}^{n-1} b_{i,n-1}(s) y_i.$$

Notice that $B(s) = B_r(1-s)$, where $B_r(s)$ is the Bézier curve defined from the control points in reverse order:

$$B_r(s) = \sum_{i=0}^{n-1} Z_{n-1-i} b_{i,n-1}(s).$$

Let s be a real number. Then the $n \times n$ lower triangular matrix $B_n^e(s)$ such that $\left(B_n^e\right)_{ij}(s) = b_{j-1,i-1}(s)$ for each $n \geq i \geq j \geq 1$ is called a Bernstein matrix [1].

Paul de Casteljau developed a very stable algorithm to evaluate Bézier curves. In this well-known Casteljau's algorithm, the number of arithmetic operations grows quadratically with n. Thus, $n(n-1)/2$ additions and $n(n-1)$ multiplications are required to calculate a point (which is not an endpoint) on a Bézier curve of degree $n-1$ [3]. A natural question to ask is if there could be a less expensive algorithm to compute a Bézier curve, which is answered, e.g., in [5] for $n \geq 7$. Here, we expand this answer for $n \leq 60$ by a different approach, which arises from the expression of an $n \times n$ Bernstein matrix in terms of the lower triangular Pascal matrix P_n (see [1]), as follows:

$$x(s) = e_n^T P_n G(-s) P_n G(-1) x, \quad y(s) = e_n^T P_n G(-s) P_n G(-1) y.$$

Here, $x = (x_0\, x_1 \cdots x_{n-1})^T$ and $y = (y_0\, y_1 \cdots y_{n-1})^T$. Furthermore, e_n denotes the nth canonical vector of $\mathbb{R}^n$, $G(s)$ is the diagonal matrix such that $e_k^T G(s) e_k = s^{k-1}$ for all $k \in \{1, ..., n\}$, and P_n is defined by

$$\left(P_n\right)_{ij} = \begin{cases} \binom{i-1}{j-1} & \text{, for } i \geqslant j; \\ 0 & \text{, otherwise.} \end{cases}$$

From now on, the methods we introduce here, which originated from the above expression, will be called Pascal matrix methods. One central problem in these

methods is to determine how we can accurately compute a matrix-vector multiplication with an $n \times n$ lower triangular Pascal matrix. In §2, we present an algorithm that utilizes only $n(n-1)/2$ additions and that computes a matrix-vector multiplication with P_n in a very precise way. It is based on the fact that a lower triangular Pascal matrix is a product of bidiagonal matrices. However, complexity can be reduced to $\mathcal{O}(n \log n)$ because the lower triangular Pascal matrix is similar to lower triangular Toeplitz matrices via diagonal matrices [6]. This transformation depends on a positive real parameter t, which is arbitrary. In [6], it is suggested that $(n-1)/e$ could be taken as a good approximate value for the optimum parameter. Here we see that, apart from a set of integer numbers that satisfy a specific property, the optimum value can be calculated. We do not yet know if the set of integers for which the optimum value does not exist is finite (so this is left as an open problem). At the end of §2, we present the results of several tests that compared the accuracy of matrix-vector multiplications carried out by a fast multiplication algorithm for both approximate and exact optimum values.

The other central problem in Pascal matrix methods is, once $z = P_n G(-1)x$ (or $z = P_n G(-1)y$) has been calculated, how to efficiently evaluate the expression $e_n^T P_n G(-s)z$? To that end, in §3, some results obtained with the use of the $\mathcal{O}(n \log n)$ Pascal matrix-vector multiplication algorithm, coupled with a Horner-type scheme in the computation of Bézier curves of degree $n-1$, are presented. When n increases, the fast matrix-vector multiplication becomes unstable, as does the evaluation of $B(s)$ for s close to 1. So, the matrix-vector multiplication must be done in a more precise way.

Thus, a stabilizing procedure to evaluate $B(s)$ should be attempted. Here, we divide the evaluation process into two parts: first, we compute $B(s)$ for $s \in [0, 1/2)$; second, we compute $B_r(s)$ for $s \in [0, 1/2]$, where $B_r(s)$ is the reverse Bézier curve, that is, $B_r(s)$ is the Bézier curve defined from the points P_{n-1}, ..., P_1 and P_0, in this order. For $n > 32$, even the 2-step $B(s)$-evaluation becomes unstable, yielding incorrect values for s close to $1/2$. In order to improve that evaluation, we introduce an affine transform of the vectors of coordinates.

2 Pascal matrix-vector multiplication

In this section, some algorithms of Pascal-type matrix-vector multiplication are discussed from the point of view of time complexity. These algorithms are founded on algebraic properties of these matrices, which will also be presented in the following sections.

2.1 $\mathcal{O}(n^2)$ arithmetic operations methods

Algorithms of matrix-vector multiplications with P_n, $P_nG_n(t)$ and $B_n^e(t)$, respectively, are presented here in the form of MATLAB functions. All of them demand $\mathcal{O}(n^2)$ arithmetic operations. We begin by observing that

$$\begin{pmatrix} 1&0&0&0\\ 1&1&0&0\\ 1&2&1&0\\ 1&3&3&1 \end{pmatrix} = \begin{pmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&1&1&0\\ 0&1&2&1 \end{pmatrix} . \begin{pmatrix} 1&0&0&0\\ 1&1&0&0\\ 0&1&1&0\\ 0&0&1&1 \end{pmatrix} .$$

Hence,

$$P_4 = \begin{pmatrix} 1 & 0\\ 0 & P_3 \end{pmatrix} E_1 = \begin{pmatrix} I_2 & 0\\ 0 & P_2 \end{pmatrix} E_2.E_1 = E_3.E_2.E_1.$$

Since for $1 < j < i \leq n$,

$$e_{i-1}^T P_{n-1} E_{n-1} e_{j-1} = e_{i-1}^T P_{n-1} \left(e_{j-1} + e_j\right) = e_{i-1}^T P_{n-1} e_{j-1} + e_{i-1}^T P_{n-1} e_j =$$

$$= \binom{i-2}{j-2} + \binom{i-1}{j-1} = \binom{i-1}{j-1} = (P_n)_{ij} ,$$

we conclude that

$$P_n = \begin{pmatrix} 1 & 0\\ 0 & P_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0\\ e_1^{(n-1)} & E_{n-1} \end{pmatrix} .$$

Therefore, the following statement has just been proved by induction:

Proposition 1 *Let P_n be the $n \times n$ lower triangular Pascal matrix. Then $P_n = E_{n-1}.\ldots.E_1$, where $E_k = I + e_{k+1}e_k^T + \ldots + e_n e_{n-1}^T$, for all $k \in \{1, \ldots, n-1\}$.*

The first algorithm, which is described as a MATLAB function, is the following:

```
function x = pascal_product(x)
%PASCAL_PRODUCT Multiply a vector x by the lower triangular
%Pascal matrix
n= length(x);
x = x(:);
for k = 2:n
    for s = n:-1:k
        x(s) = x(s) + x(s-1);
    end
```

```
end
```

Notice that, from the above factorization, we can conclude that

$$P_n = \begin{pmatrix} I_k & 0 \\ 0 & P_{n-k} \end{pmatrix} \begin{pmatrix} P_k & 0 \\ Z & W \end{pmatrix},$$

where

$$\begin{pmatrix} Z & W \end{pmatrix} = \begin{pmatrix} \binom{k}{0} & \binom{k}{1} & \dots & \binom{k}{k} & 0 & \dots & 0 \\ 0 & \binom{k}{0} & \binom{k}{1} & \dots & \binom{k}{k} & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \dots & \ddots & \vdots \\ 0 & \dots & 0 & \binom{k}{0} & \binom{k}{1} & \dots & \binom{k}{k} \end{pmatrix}.$$

A similar factorization can be done on $P_n G_n(t)$:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \binom{1}{1}t & 0 & \dots & 0 \\ 1 & \binom{2}{1}t & \binom{2}{2}t^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \binom{n-1}{1}t & \binom{n-1}{2}t^2 & \dots & \binom{n-1}{n-1}t^{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 1 & t \end{pmatrix} \dots \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & t & 0 & \dots & 0 \\ 0 & 1 & t & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & t \end{pmatrix}.$$

As a result, we can state our next proposition with no further proof:

Proposition 2 *Let P_n be the $n \times n$ lower triangular Pascal matrix and let $G(t) = diag(1, t, t^2, ..., t^{n-1})$. Then $P_n G_n(t) = E_{n-1}(t)...E_1(t)$ where, for $1 \leq k \leq n-1$,*

$$E_k(t) = e_1 e_1^T + ... + e_k e_k^T + e_{k+1}[e_k + te_{k+1}]^T + ... + e_n[e_{n-1} + te_n]^T.$$

The second algorithm, which utilizes Proposition 2, is defined as follows:

```
function x = pascal_g_product(x,t)
%PASCAL_G_PRODUCT Multiply a vector x by PG(t) where
%  P is the lower triangular Pascal matrix and
%  G(t) = diag(1,t,t^2,...,t^{n-1})
n= length(x);
x = x(:);
for k = 2:n
```

```
    for s = n:-1:k
        x(s) = x(s-1) + t*x(s);
    end
end
```

By using the notation of [4], a Bernstein matrix can be described as

$$B_n^e(t) = P_n[1-t]G_n(t),$$

where $(P_n[t])_{ij} = (P_n)_{ij}\, t^{i-j}$, $1 \leq j \leq i \leq n$. Therefore, it is not difficult to conclude that a Bernstein matrix has the following bidiagonal factorization:

$$\begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 1-t & \binom{1}{1}t & 0 & ... & 0 \\ (1-t)^2 & \binom{2}{1}(1-t)t & \binom{2}{2}t^2 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ (1-t)^{n-1} & \binom{n-1}{1}(1-t)^{n-2}t & \binom{n-1}{2}(1-t)^{n-3}t^2 & ... & \binom{n-1}{n-1}t^{n-1} \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & 0 & ... & 0 & 0 \\ 0 & 1 & ... & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \\ 0 & 0 & ... & 1 & 0 \\ 0 & 0 & ... & 1-t & t \end{pmatrix} ... \begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 1-t & t & 0 & ... & 0 \\ 0 & 1-t & t & ... & 0 \\ \vdots & \vdots & \ddots & \ddots & \\ 0 & 0 & ... & 1-t & t \end{pmatrix}.$$

This fact leads us to the following statement:

Proposition 3 *Let $B_n^e(t)$ be an $n \times n$ Bernstein matrix. Then*

$$B_n^e(t) = E_{n-1}^e(t)...E_1^e(t) \text{ where, for } 1 \leq k \leq n-1,$$

$$E_k^e(t) = e_1 e_1^T + ... + e_k e_k^T + e_{k+1}[(1-t)e_k + te_{k+1}]^T + ... + e_n[(1-t)e_{n-1} + te_n]^T.$$

Proposition 3 yields the following algorithm, which is essentially Casteljau's.

```
function x = bernstein_product(x,t)
%BERNSTEIN_PRODUCT Multiply a vector x by a Bernstein matrix
n= length(x);
x = x(:);
t1 = 1-t;
for k = 2:n
```

```
    for s = n:-1:k
        x(s) = t1*x(s-1) + t*x(s);
    end
end
```

Similarly, we can conclude that

$$B_n^e(t) = \begin{pmatrix} I_k & 0 \\ 0 & B_{n-k}^e(t) \end{pmatrix} \begin{pmatrix} B_k^e(t) & 0 \\ Z & W \end{pmatrix},$$

where $\begin{pmatrix} Z & W \end{pmatrix}$ is the matrix

$$\begin{pmatrix} (1-t)^k & \binom{k}{1}(1-t)^{k-1}t & \dots & \binom{k}{k}t^k & 0 & \dots & 0 \\ 0 & (1-t)^k & \binom{k}{1}(1-t)^{k-1}t & \dots & \binom{k}{k}t^k & \dots & 0 \\ \vdots & \ddots & \ddots & \dots & & \ddots & \vdots \\ 0 & \dots & (1-t)^k & \binom{k}{1}(1-t)^{k-1}t & \dots & & \binom{k}{k}t^k \end{pmatrix}.$$

2.2 Scaling a lower triangular Pascal matrix

Let $Z_0 = (x_0, y_0)$, $Z_1 = (x_1, y_1)$, ..., $Z_{n-1} = (x_{n-1}, y_{n-1})$ be n points of $\mathbb{R}^2$. Let $x = (x_0\, x_1\, \dots\, x_{n-1})^T$ and $y = (y_0\, y_1\, \dots\, y_{n-1})^T$. For $s \in [0,1]$, let $B_n^e(s)$ be the $n \times n$ Bernstein matrix. Then a spectral decomposition of $B_n^e(s)$ is $B_n^e(s) = P_n G_n(s) P_n^{-1}$, where P_n is the $n \times n$ lower triangular Pascal matrix and $G_n(s) = diag(1, s, ..., s^{n-1})$ ([1]). Since $P_n^{-1} = G_n(-1) P_n G_n(-1)$ ([1]), we conclude that the coordinates of the Bézier curve $B(s)$, $s \in [0,1]$, which were defined from Z_0, ..., Z_{n-1} (in this order), are given by

$$x(s) = e_n^T P_n G_n(-s) P_n G_n(-1) x,\ y(s) = e_n^T P_n G_n(-s) P_n G_n(-1) y. \quad (1)$$

The computation of a matrix-vector multiplication with the Pascal matrix P_n can be carried out in $\mathcal{O}(n \log n)$ operations [6]. Hence, for each $s \in [0,1]$, the computation of both $x(s)$ and $y(s)$ also requires $\mathcal{O}(n \log n)$ operations. However, arithmetic operations with Pascal matrices are very unstable because of the different magnitudes of their entries. In this section, a preconditioning technique is proposed to deal with this instability. This technique, which was introduced in [6], is based on the factorization

$$P_n = D_n(t)\, T_n(t)\, D_n(t)^{-1}, \quad (2)$$

where $D_n(t) = diag(0!, \frac{1!}{t}, \ldots, \frac{(n-1)!}{t^{n-1}})$, and

$$T_n(t) = \begin{pmatrix} 1 & 0 & 0 & \ldots 0 \\ \frac{t}{1!} & 1 & 0 & \ldots 0 \\ \frac{t^2}{2!} & \frac{t}{1!} & 1 & \ldots 0 \\ \vdots & \vdots & \vdots & \ddots \vdots \\ \frac{t^{n-1}}{(n-1)!} & \frac{t^{n-2}}{(n-2)!} & \frac{t^{n-3}}{(n-3)!} & \ldots 1 \end{pmatrix}.$$

Note that, for $0 < t \leq 1$, the entries of the Toeplitz matrix $T_n(t)$ vary from $\frac{t^{n-1}}{(n-1)!}$ to 1. That is, the magnitudes of the entries of $T_n(t)$ for $0 < t \leq 1$ are quite different as well. In order to effectively compute matrix-vector multiplications with a Pascal matrix via the decomposition given in (2), we need to find a scaling parameter t for which the magnitudes of the elements of $T_n(t)$ are as close as possible to one another. For this purpose, since the entries of the matrix $T_n(t)$ are of the form

$$f(m) = \frac{t^m}{m!},$$

where $m = 0, 1, \ldots, n-1$, we are going to search for a positive value t that minimizes $\max f / \min f$. Observe that

$$\frac{\max\limits_{0 \leq m \leq n-1} \frac{t^m}{m!}}{\min\limits_{0 \leq m \leq n-1} \frac{t^m}{m!}} = \frac{\max\limits_{i,j} \left(T_n(t)\right)_{ij}}{\min\limits_{i,j} \left(T_n(t)\right)_{ij}} = \frac{\max\limits_{i} \left(D_n(t)^{-1}\right)_{ii}}{\min\limits_{i} \left(D_n(t)^{-1}\right)_{ii}} = \frac{\max\limits_{i} \left(D_n(t)\right)_{ii}}{\min\limits_{i} \left(D_n(t)\right)_{ii}}.$$

The following lemma proves basic facts about the monotonicity of f.

Lemma 4 *Let $t \in (0, \infty)$. Consider the function $f : \mathbb{N} \to [0, \infty)$ defined by $f(m) = \frac{t^m}{m!}$. Then*

(i) f is a nondecreasing function for integers m such that $m \leqslant \lfloor t \rfloor \leqslant t$;
(ii) f is a nonincreasing function for integers m such that $t \leqslant \lceil t \rceil \leqslant m$.

PROOF.

(i) $f(m-1) = \frac{t^{m-1}}{(m-1)!} = \frac{m}{m} \frac{t^{m-1}}{(m-1)!} \leqslant \frac{t^m}{m!} = f(m)$, since $m \leqslant t$.

(ii) $f(m-1) = \frac{t^{m-1}}{(m-1)!} = \frac{m}{m} \frac{t^{m-1}}{(m-1)!} \geqslant \frac{t^m}{m!} = f(m)$, for $m \geqslant t$.

□

Therefore, if the optimum value of t exists, it will belong to the interval $[1, n-1)$. Now, $\max f / \min f = \dfrac{t^{\lfloor t \rfloor}}{(\lfloor t \rfloor)!}$, if $0 \leq m \leq t$; if $t < m \leq n-1$, then $\max f / \min f = \dfrac{t^{\lceil t \rceil}(n-1)!}{t^{n-1}(\lceil t \rceil)!}$. Thus, if the optimum value exists, it will correspond to the parameter $t \in [1, n-1)$ for which

$$\min \max \{ \frac{t^{\lfloor t \rfloor}}{(\lfloor t \rfloor)!}, \frac{t^{\lceil t \rceil}(n-1)!}{t^{n-1}(\lceil t \rceil)!} \}$$

is achieved.

Lemma 5 *Let n be an integer greater than 1. Let f_1 and f_2 be two functions defined on the subset $(0, n-1)$ of the real numbers such that $f_1(t) = \dfrac{t^{\lfloor t \rfloor}}{(\lfloor t \rfloor)!}$ and $f_2(t) = \dfrac{t^{\lceil t \rceil}(n-1)!}{t^{n-1}(\lceil t \rceil)!}$. Then*

(i) f_1 is an increasing continuous function;
(ii) f_2 is a nonincreasing function and it is continuous except at integer values.

Remark 6 *Therefore, if there is a $\tilde{t} \in (0, n-1)$ such that $f_1(\tilde{t}) = f_2(\tilde{t})$, this is the optimum value.*

Lemma 7 *If there is a $\tilde{t} \in (k, k+1)$ such that $f_1(\tilde{t}) = f_2(\tilde{t})$, then*

$$\tilde{t} = \sqrt[n-2]{\frac{(n-1)!}{k+1}}.$$

PROOF. Let $\tilde{t}$ be such that $f_1(\tilde{t}) = f_2(\tilde{t})$. Since $\tilde{t} \in (k, k+1)$, it follows that

$$\frac{\tilde{t}^k}{k!} = \frac{\tilde{t}^{k+1}\,(n-1)!}{(k+1)!\,\tilde{t}^{n-1}} \Rightarrow \frac{k+1}{(n-1)!} = \frac{1}{\tilde{t}^{n-2}} \Rightarrow \tilde{t} = \sqrt[n-2]{\frac{(n-1)!}{k+1}}.$$

□

Lemma 8 *If there is a $\tilde{t} \in (k, k+1)$ such that $f_1(\tilde{t}) = f_2(\tilde{t})$, then $k = \lfloor \sqrt[n-1]{(n-1)!} \rfloor$.*

PROOF. From the former lemma, if $k < \tilde{t} < k+1$ then $\tilde{t}^{n-2} = \dfrac{(n-1)!}{k+1}$. Hence,

$$k^{n-2} < \frac{(n-1)!}{k+1} < (k+1)^{n-2}.$$

But this yields

$$(n-1)! < (k+1)^{n-1} \text{ and } (n-1)! > (k+1)k^{n-2} > k^{n-1}.$$

Therefore,

$$k = \max\{m \in \mathbb{Z}^+ | \, m^{n-1} < (n-1)!\} = max\{m \in \mathbb{Z}^+ | \, m < \sqrt[n-1]{(n-1)!}\}$$

That is, $k = \lfloor \sqrt[n-1]{(n-1)!} \rfloor$. $\square$

Remark 9 *Note that, for $n = 3$, $f_1(1) = 1$, $f_2(1) = 2$, $f_2(1^+) = 1$. So, in this case, there is no optimum value because $f_2(1) > f_1(1) \geq f_2(1^+)$. Thus, a number greater than 1 but very close to it can be considered a good value for scaling the 3×3 Pascal matrix. Notice that there is a positive integer n, $n > 3$, such that for some integer k, $1 \leq k \leq n-2$, $f_2(k) > f_1(k) \geq f_2(k^+)$ if and only if*

$$\frac{k^k(n-1)!}{k!\, k^{n-1}} > \frac{k^k}{k!} \geq \frac{k^{k+1}(n-1)!}{(k+1)!\, k^{n-1}},$$

that is, if and only if

$$k^{n-1} < (n-1)! \text{ and } k^{n-1} + k^{n-2} \geq (n-1)!.$$

Proposition 10 *If n is an integer number, $n > 3$, such that $k^{n-1} < (n-1)!$ and if $k^{n-1} + k^{n-2} \geq (n-1)!$ for some integer k, $1 \leq k \leq n-2$, then*

$$\frac{n-1}{e} < k < \frac{n-1}{2}.$$

PROOF. If $k \geq (n-1)/2$, then

$$k^{n-1} \geq (\frac{n-1}{2})^{n-1} = \frac{(n-1)^{n-1}}{2^{n-1}}.$$

But, if $n \geq 1$, then $n^n \geq (n!)^2$. Thus, we have

$$\frac{(n-1)^{n-1}}{2^{n-1}} \geq \frac{((n-1)!)^2}{2^{n-1}}.$$

Now, for $n > 3$, $2^{n-1} < (n-1)!$. Hence, $k^{n-1} > (n-1)!$.

On the other hand, k must be greater than $(n-1)/e$, because

$$k \leq \frac{n-1}{e} \Longrightarrow k^{n-1} + k^{n-2} \leq (\frac{n-1}{e})^{n-1}[1 + \frac{e}{n-1}]$$

$$< \sqrt{2\pi(n-1)}(\frac{n-1}{e})^{n-1} < (n-1)!,$$

by the Stirling formula. □

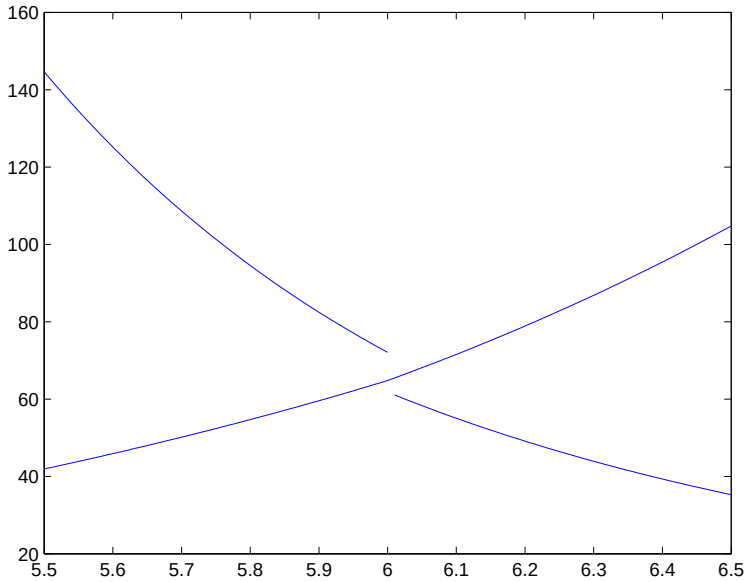

Fig. 1. $n = 15$: one can see that f_1 and f_2 don't intersect

Remark 11 *As far as we know, the conjecture about the finiteness of the set of integers n such that $k^{n-1} < (n-1)! \leq k^{n-1} + k^{n-2}$ for some integer k, where $1 \leq k \leq n-2$, has not yet been proved. Each integer n less than 10000 belonging to this set is displayed in Table 1, together with its corresponding integer k, $1 \leq k \leq n-2$.*

n	15	39	74	527	3171	5908	7036	7534	7537
k	6	15	28	195	1168	2175	2590	3194	3401

Table 1
$k^{n-1} \leq (n-1)! \leq k^{n-1} + k^{n-2}$

Remark 12 *For each value of n in Table 1, there is no optimum value and we will take $t = k_n$ as our scaling parameter, where k_n is the value of k corresponding to n. For the other values of n, $4 \leq n \leq 10000$, the optimum value is given by*

$$t_{opt} = \sqrt[n-2]{\frac{(n-1)!}{k+1}}$$

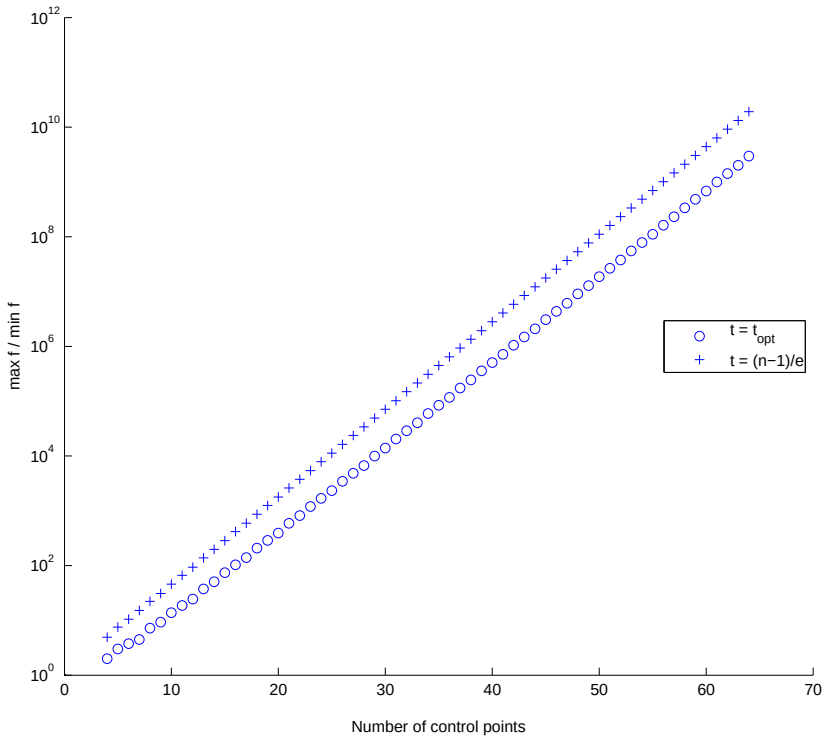

Fig. 2. $\max f / \min f$, where $f(k) = t^k/k!$, $0 \leq k \leq n-1$

where $k = \lfloor \sqrt[n-1]{(n-1)!} \rfloor$. *Therefore, from Lemma 7,* $k = \lfloor t_{opt} \rfloor$, *and since* $k+1 > \sqrt[n-1]{(n-1)!}$,

$$k < t_{opt} < \sqrt[n-2]{\frac{(n-1)!}{\sqrt[n-1]{(n-1)!}}} = \sqrt[n-1]{(n-1)!} < k+1.$$

We see in Figure 2.2 that the value of $\frac{\max_{0\leq k\leq n-1} \frac{t^k}{k!}}{\min_{0\leq k\leq n-1} \frac{t^k}{k!}}$ *at* $t = (n-1)/e$ *is greater than that at* $t = t_{opt}$, *for all* $n \in \{4, 5, ..., 64\}$.

Numerical experiments have shown that a Pascal matrix-vector multiplication via $T_n(t)$ performs better in a narrow range of scaling parameters t that includes t_{opt}. For $n = 32$, the Figure 2.2 shows a logarithmic scale graph of various computed values of the error $E_n(t) = ||e_1 - D_n(t).T_n(t).D_n(t)^{-1}z||_2$ versus the scaling parameter t, where e_1 is the first canonical vector of $\mathbb{R}^n$ and z is the vector defined by $z_k = (-1)^k$, $k = 1, ..., n$. The scaling parameters have been taken from a set that includes twenty points linearly spaced between and including $t_n = (n-1)/e$ and $T_n = t_{opt}$. The other points of the set have been taken from t_n and T_n with step sizes of -0.05 and 0.05, respectively. We see in this graph that the errors are very sensitive to small variations of the scaling parameter and that this sensitivity increases when the scaling param-

eter moves away from the interval $[t_n, T_n]$. In the graph, we also note that the error obtained by using the scaling parameter t_n is slightly less than the one obtained by using T_n. This also occurs with most of the numbers n between 4 and 64, as we can observe in Figure 2.2, which shows a logarithmic scale graph of $E_n(t_n)$ and $E_n(T_n)$, both versus the number (n) of the control points of a Bézier curve. On the other hand, the result obtained by the multiplication $P_n z$ via the *pascal_product* function matches with e_1 for each n between 4 and 64.

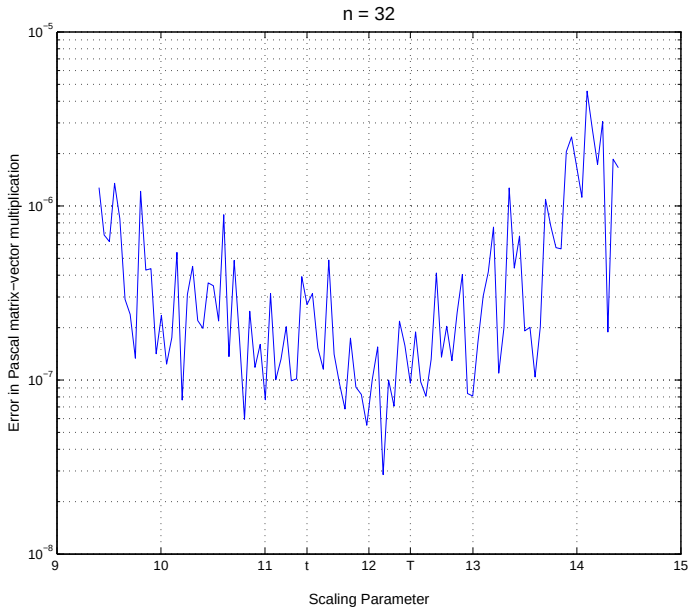

Fig. 3. Error $E_n(t) = ||e_1 - D_n(t).T_n(t).D_n(t)^{-1}z||_2$

3 On computing a Bézier curve

In this section, we are going to compute a Bézier curve. First, notice that if the control points are translated by a vector $v = (p, q)$, the Bézier curve is also translated by v. Another feature of a Bézier curve is that a uniform scaling of the control points yields a uniform scaling of the curve. Hence, we assume without loss of generality that all of the coordinates of the control points are both positive and less than or equal to 1.

3.1 $B(s)$-evaluation

Pascal matrix methods can be used to compute a Bézier curve $B(s)$ of degree $n-1$ via the decomposition $B_n^e(s) = P_n G_n(-s) P_n G_n(-1)$ in the following way:

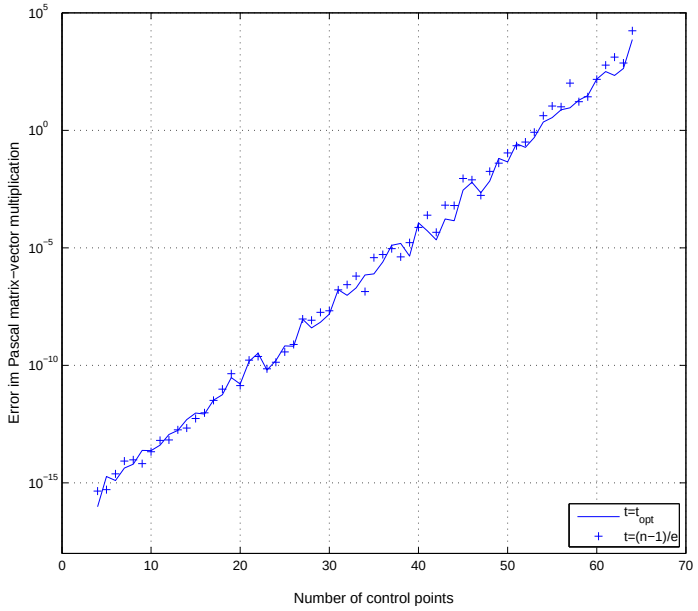

Fig. 4. Error $E_n(t)$, $4 \leq n \leq 64$

first, $z = P_n G_n(-1)x = P_n x_-$ and $w = P_n G_n(-1)y = P_n y_-$ are computed; then the polynomials $e_n^T P_n G_n(-s)z$ and $e_n^T P_n G_n(-s)w$ are evaluated. When n is small, e.g., $n = 32$, z and w have 2-norms around 10^9 for $||x||_\infty = ||y||_\infty = 1$. In this case, both polynomials could be efficiently evaluated for each $s \in [0, 1]$. The function *pascal_g_product* does the evaluation, with $n(n-1)/2$ additions and $n(n-1)/2$ multiplications for each s. A less expensive alternative is to use a Horner-like scheme that evaluates the polynomial concomitantly with the binomial coefficients.

The control points for the tests were defined by the MATLAB command $rand(n, 2)$, which returns an $n \times 2$ matrix containing pseudo-random values drawn from a uniform distribution on the unit interval. We have used the Pas-

n	$\|\|B_P - B_C\|\|_\infty$	$time_P$	$time_C$
4	7.7716e-16	0.001810	0.001273
8	2.8547e-14	0.001954	0.002164
15	9.3585e-11	0.002095	0.005024
16	1.9592e-10	0.002238	0.005475
24	1.2341e-06	0.002518	0.009927
32	0.0190	0.003169	0.015896

Table 2
The Pascal matrix method vs. Casteljau's method

cal matrix-vector multiplication performed from the similar Toeplitz matrix $T(t)$, with t having been found by our procedure (according to our calculations in the last section) and the $B(s)$-evaluation having been given by the Horner-like scheme cited above. The results were compared with those obtained by Casteljau's method. In table 2 the ∞-norms of the differences of the results obtained by the Pascal matrix method (B_P) and Casteljau's method (B_C) are listed. We have also listed the average computing times of each $B(s)$-evaluation ($\Delta\, s = 1/128$). Each one corresponds to the smallest time among 10 elapsed times obtained from consecutive executions of the procedure, and were computed by MATLAB's built-in tic/toc functions.

For $n = 32$, the $B(s)$-evaluation becomes unstable when s approaches 1. In order to locally and globally improve the evaluation, we have made a simple procedure that has yielded more precise results. Specifically, we divided the process of evaluation in two independent steps:

(a) evaluate $e_n^T P_n G_n(-s)z$ and $e_n^T P_n G_n(-s)w$ for $0 \leq s \leq 1/2$;
(b) evaluate $e_n^T P_n G_n(-s)z_r$ and $e_n^T P_n G_n(-s)w_r$ for $1/2 > s \geq 0$, which is equivalent to evaluating $e_n^T P_n G_n(-s)z$ and $e_n^T P_n G_n(-s)w$ for $1/2 < s \leq 1$.

The results indicate that the procedure has worked well as far as $n = 41$, and some of these results can be seen in Table 3.

n	$\|\|B_P - B_C\|\|_\infty$	$time_P$	$time_C$
32	2.3113e-07	0.005602	0.015901
36	2.6961e-05	0.006106	0.019464
39	1.3152e-04	0.005873	0.022314
41	4.8668e-04	0.006229	0.024372
42	0.0022	0.006387	0.025476
48	0.1112	0.006706	0.032366

Table 3
The Pascal matrix method with reverse evaluation vs. Casteljau's method

Two remarkable facts can be gleaned from table 3: first, the time of computation for $n = 39$ is smaller than that for $n = 41$; second, there is a sudden loss of precision when going from $n = 41$ to $n = 42$. One explanation for the first fact is that there is no calculation to find the optimum value to scale P_{39}, so 15 is taken to be the optimum value. The second fact surely has to do with the limitations of the floating point arithmetic of our machine, a 32-bit AMD Athlon XP 1700+ (1467 MHz).

3.2 On conditioning the vectors of the coordinates

One way to overcome this lack of stability is to transform the coordinate vectors x and y into a vector very close to $e^T = (1\,1\,...\,1)^T$. Since

$$v_k = \frac{1}{m+1}v_{k-1} + \frac{m}{m+1}e$$

is a stationary scheme that converges to the solution e of $Ix = e$ for any v_0, the idea is to compute the Bézier curve $T_m(B)$ from control points $W_0 = T_m(Z_0)$, ..., $W_{n-1} = T_m(Z_{n-1})$, where $T_m(v) = (v + m.e)/(m+1)$, and then to obtain B by inversely transforming the points of $T(B)$. Note that

$$(T_m(B)(s))^T = e_n^T P_n G(-s) P_n G(-1) \Big(T_m(x)\ T_m(y) \Big).$$

Hence,

$$e_n^T P_n G(-s) P_n G(-1) T_m(x) = e_n^T P_n G(-s) P_n G(-1)(x + m.e)/(m+1) =$$

$$= \frac{1}{m+1} e_n^T P_n G(-s) P_n G(-1) x + \frac{m}{m+1} e_n^T P_n G(-s) P_n G(-1) e =$$

$$= \frac{1}{m+1} x(s) + \frac{m}{m+1}.$$

Thus, since an analogous result is obtained with y, we have $x(s) = (m+1)T_m(B)(s) - m$. Observe that

$$T_m T_{m'}(v) = (v + [(m+1)(m'+1) - 1].e)\,/[(m+1)(m'+1)] = T_{(m+1)(m'+1)-1}.$$

Our strategy to obtain a better $B(s)$-evaluation is the following:

(a) evaluate $e_n^T P_n G_n(-s) z$ and $e_n^T P_n G_n(-s) w$ for $0 \le s \le 1/3$;
(b) evaluate $e_n^T P_n G_n(-s) P_n G_n(-1) T_m(x)$ and $e_n^T P_n G_n(-s) P_n G_n(-1) T_m(y)$ for $1/3 < s < 2/3$, and inversely transform them;
(c) evaluate $e_n^T P_n G_n(-s) z_r$ and $e_n^T P_n G_n(-s) w_r$ for $1/3 \ge s \ge 0$.

For the experiments we conducted, we generated the coordinates of the n control points by the commands $A = rand(n, 2)$ and $A = A/norm(A)$. Then, we applied sequences of $T'_m s$ to A for different values of n. For instance, by having defined $m + 1 = 32768$ (2^{15}), $m' + 1 = 1024$ and $m'' + 1 = 4$, we have tested the following sequences: $T_m \circ T_m$ for $n = 42, 48, 54$; $T_{m'} \circ T_m \circ T_m$ for $n = 59$; and $T_{m''} \circ T_{m'} \circ T_m \circ T_m$ for $n = 64$. The Pascal matrix vector multiplications were computed by the function *pascal_product* and the polynomial evaluation in (b), by the function *pascal_g_product*. The results are displayed in Table 4.

n	$\|\|B_P - B_C\|\|_\infty$	$time_P$	$time_C$
42	8.3290e-07	0.013525	0.025363
48	1.7620e-06	0.015566	0.032190
54	2.3903e-04	0.017786	0.039937
59	9.9235e-04	0.019782	0.047320
64	0.0048	0.022044	0.055178

Table 4
The Pascal matrix method with piecewise evaluation vs. Casteljau's method

4 Conclusions

We have presented results obtained from methods used to compute a Bézier curve of degree $n-1$, which was done for various values of n. They were obtained from methods that calculated the curve by matrix-vector multiplications with the $n \times n$ lower triangular Pascal matrix P_n, which are called Pascal matrix methods in this paper. With this in mind, we have introduced two algorithms: one demands $n(n-1)/2$ additions, and it is based on the fact that P_n is a product of bidiagonal matrices with 0 and 1; the other one, which demands $\mathcal{O}(n \log n)$ algebraic operations, depends on a positive scaling parameter in order to minimize the magnitudes of the entries of P_n when P_n is considered as a scaled Toeplitz matrix. We have seen that there is a function that relates n to that optimum value, except for some integers belonging to a certain set. We do not yet know whether this set is finite. Once the matrix-vector multiplication has been done, a polynomial evaluation should have been carried out for various values $s \in [0, 1]$. By adopting some specific strategems, such as affine transforms of the vector of coordinates of the curve, the use of Pascal matrix-vector multiplication, and polynomial evaluation, we have obtained results close to those obtained by Casteljau's method for $n \leq 60$. Moreover, the tests using all of these procedures combined have also indicated that they are more effective with respect to the time of computation than Casteljau's algorithm, at least for $n \leq 60$.

References

[1] L. Aceto, D. Trigiante, The matrices of Pascal and other greats, Amer. Math. Monthly 108 (2001), 232–245.

[2] P. Bézier, Numerical Control: Mathematics and Applications, John Wiley & Sons (London, 1972).

[3] W. Boehm, A. Müller, On de Casteljau's algorithm, Comput. Aided Geom. D. 16 (1999), 587–605.

[4] G. S. Call, D. J. Velleman, Pascal's Matrices, Amer. Math. Monthly 100 (1993), 372–376.

[5] H. N. Phien, N. Dejdumrong, Efficient algorithms for Bézier curves, Comput. Aided Geom. D. 17 (2000), 247–250.

[6] X. Wang, J. Zhou, A fast eigenvalue algorithm for Pascal Matrices, Appl. Math. Comput. 183 (2006), 713–716.