*Mathematical and Computational Applications*

MDPI

*Article*

# A Computational Method with Maple for Finding the Maximum Curvature of a Bézier-Spline Curve

**Henk Pijls [1] and Le Phuong Quan [2,*]**

[1] Korteweg-de Vries Institute for Mathematics, University of Amsterdam, Science Park 105-107, 3rd Floor (Entrance via Nikhef), 1098 XG Amsterdam, The Netherlands
[2] Department of Mathematics, College of Natural Sciences, Cantho University, 3/2 Street, Cantho City 900000, Vietnam
* Correspondence: lpquan@ctu.edu.vn; Tel.: +84-76-290-2215; Fax: +84-07103-832062

**Abstract:** In this paper, we propose two Maple procedures and some related utilities to determine the maximum curvature of a cubic Bézier-spline curve that interpolates an ordered set of points in $\mathbb{R}^2$ or $\mathbb{R}^3$. The procedures are designed from closed-form formulas for such open and closed curves.

**Keywords:** approximation; Bézier-spline curve; computer algebra system; Frenet frame; interpolation; parametrization

**MSC:** 41A15; 53A04; 65D05; 65D17; 68W30



**Citation:** Pijls, H.; Quan, L.P. A Computational Method with Maple for Finding the Maximum Curvature of a Bézier-Spline Curve. *Math. Comput. Appl.* **2023**, *28*, 56. https://doi.org/10.3390/mca28020056

Academic Editor: Maria Amélia Ramos Loja

Received: 3 March 2023
Revised: 30 March 2023
Accepted: 3 April 2023
Published: 8 April 2023



## 1. Introduction

We recall here the definition of *curvature* of a smooth curve. This is a fundamental concept in differential geometry that has been studied deeply in applied mathematics, engineering, and computer graphics. The problem of finding the curvature extremum has been investigated by many authors. We can point out some of their results on establishing necessary and sufficient conditions for the regularity of offset curves or tubular surfaces, designing various types of aesthetic curves from constrained conditions, representing and modifying curves to adapt principles of interpolation and animation, etc. In the following, we give a short description of the method for solving this problem and its limitations.

If $\mathbf{r}\colon [a,b] \longrightarrow \mathbb{R}^3$ is the position vector of a smooth curve $\mathscr{L}$, then the point $\mathbf{r}(t)$ of $\mathscr{L}$ at $t \in [a,b]$ is written as a 3-tuple $(x(t),y(t),z(t))$, or a column vector $\langle x(t),y(t),z(t)\rangle = (x(t),y(t),z(t))^T$ when used in a matrix expression; hence $\mathbf{r}(t)^T$ is the row matrix $[x(t) \quad y(t) \quad z(t)]$. The image of $\mathbf{r}$ or the map $\mathbf{r}$ itself is called a parametrization of $\mathscr{L}$. Along $\mathscr{L}$, we consider the coordinates of the Frenet frame $\{\hat{\mathbf{T}},\hat{\mathbf{N}},\hat{\mathbf{B}}\}$ whose origin is located at points of $\mathscr{L}$. From differential geometry and calculus (see [1,2]), we know that

$$\hat{\mathbf{T}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}, \quad \hat{\mathbf{B}} = \frac{\mathbf{v}\times\mathbf{a}}{\|\mathbf{v}\times\mathbf{a}\|}, \quad \hat{\mathbf{N}} = \hat{\mathbf{B}}\times\hat{\mathbf{T}},$$

where $\mathbf{v}(t) = \mathbf{r}'(t) = (x'(t),y'(t),z'(t))^T$ and $\mathbf{a}(t) = \mathbf{r}''(t) = (x''(t),y''(t),z''(t))^T$ are usually called the velocity vector and the acceleration vector of $\mathscr{L}$ at $t$ (or at $\mathbf{r}(t)$).

Let $\mathbf{r} = \mathbf{r}(t) = (x(t),y(t),z(t))^T$ be a parametrization of a smooth space curve $\mathscr{L}$ with $\mathbf{a} = \mathbf{r}'' \in C([a,b])$. Then, the curvature of $\mathscr{L}$ at $t$ is given by

$$\kappa = \frac{\|\mathbf{v}\times\mathbf{a}\|}{v^3}, \tag{1}$$

where $\mathbf{v} = \mathbf{v}(t) = \mathbf{r}'(t)$, $\mathbf{a} = \mathbf{a}(t) = \mathbf{r}''(t)$, and $v = \|\mathbf{v}(t)\|$. Under this assumption, we usually find the maximum value of $\kappa(t)$ on $[a,b]$ by choosing the largest value of $\kappa$ at the points where $\kappa'(t) = 0$. However, how to solve the equation $\kappa'(t) = 0$ exactly or

approximately? In general, we cannot do it. Moreover, apart from this, the expression for $\kappa'(t)$ is complicated! Let us take $K = \kappa^2$ and write it out in the components of $\mathbf{v} \times \mathbf{a}$. This gives

$$K = \frac{(y'z'' - z'y'')^2 + (z'x'' - x'z'')^2 + (x'y'' - y'x'')^2}{[(x')^2 + (y')^2 + (z')^2]^3}.$$

On the other hand, if $\mathbf{r} = \mathbf{r}(s)$ is the parametrization of $\mathscr{L}$ by the arc length $s$, then from the relation $\hat{\mathbf{T}}'(s) = \kappa(s)\hat{\mathbf{N}}(s)$ we obtain

$$\hat{\mathbf{T}}''(s) = \kappa'(s)\hat{\mathbf{N}}(s) + \kappa(s)\hat{\mathbf{N}}'(s).$$

This gives a simple expression for $\kappa'(s)$:

$$\hat{\mathbf{T}}''(s) \cdot \hat{\mathbf{N}}(s) = \kappa'(s).$$

However, we again encounter another hard problem: how to convert a parametrization of a smooth space curve $\mathscr{L}$ by a general parameter $t$ into the parametrization by the arc length $s$. In general, this is impossible.

To overcome those obstacles, many researchers restricted their attention to the class of Bézier curves and their variants. Recently, the papers related to maximizing or minimizing the curvature of these curves provided a lot of theoretical results and useful algorithms, as well as practical tools. We highlight some representative papers with a brief note. Ref. [3] presents a unique design on a piecewise quadratic Bézier curve that interpolates its local maximum curvature points that are also its control points. Ref. [4], adopted from [3], proposes new methods to modify local curvature at the interpolation points by taking basis functions of higher degree. Ref. [5] provides conditions for the curvature of a quadratic rational Bézier curve to be monotone or to have a local minimum and maximum. Ref. [6] establishes conditions for Bézier plane curves generated by a matrix to have monotone curvature. Ref. [7] also establishes conditions for Bézier curves to have monotone curvature, based on control points of the position vector of the curve and its derivatives. Ref. [8] treats typical Bézier plane curves with one curvature extremum that can be easily calculated, which can help to divide the curve into two typical curves with monotone curvature.

Traditionally, the papers noted above and many others paid much attention to control points and polygons. Actually, these objects have direct effects on the shape of the curves, so they have been modified in order to obtain a curve with properties needed in design applications. However, we have some changes in mind when relating this widespread trend to our result in [9]. The formula in [9] (Theorem 3.1) can be seen as a way of approximation by interpolation with Bézier-spline curves. Therefore, we prefer to place emphasis on interpolated points. These points can be chosen in a way to design curves in $\mathbb{R}^2$ or $\mathbb{R}^3$ with desired shapes or can be taken from special partitions of the parameter interval of a smooth curve with given parametrization to approximate its curvature extremum.

Now, we go back to our main purpose: making a Maple procedure to compute the maximum value $\kappa_{\max}$ of the curvature. We restrict our attention to *Bézier-spline curves*. This objective is based on the power of Maple on symbolic computation and on solving polynomial equations of high degree and on the explicit piecewise cubic parametrization of these special curves.

The present paper is organized as follows. In Section 2, we construct the Maple procedures to represent Bézier-spline space curves for both open and closed curves. In Section 3, we propose a pseudo-algorithm for computing $\kappa_{\max}$, then we provide the full code of the procedures corresponding to the algorithm. In Section 4, we discuss some modifications to obtain procedures to represent Bézier-spline plane curves and to compute their maximum curvature. In Section 5, we give some concluding remarks.

## 2. Bézier-Spline Space Curves with Maple Parametrization

In this section, we consider a Bézier-spline curve, which is obtained from a closed-form solution to the inverse problem, which interpolates an ordered set of points $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n$, given in [9]. Such a plane curve can be obviously extended to a space curve $\mathscr{C}$ given by a piecewise cubic function $\mathbf{f}$ of $C^2([0,n])$. According to the construction of such curves in [9], we present here a more convenient way to derive their parametrization.

First, $\mathbf{f}(t)$ is composed of the cubic functions $\mathbf{f}_k$ given by

$$\mathbf{f}_k(t) = (1-t)^3\mathbf{s}_{k-1} + 3t(1-t)^2\mathbf{p}_{k-1} + 3t^2(1-t)\mathbf{q}_k + t^3\mathbf{s}_k, \tag{2}$$

where $k = 1, \ldots, n$ and $t \in [0,1]$, and $\mathbf{f}_k$ is the parametrization of the cubic Bézier curve $\mathscr{C}_k$ with the control points $\mathbf{s}_{k-1}$, $\mathbf{p}_{k-1}$, $\mathbf{q}_k$, and $\mathbf{s}_k$. These points satisfy the known relations

$$\mathbf{p}_{k-1} = \frac{2}{3}\mathbf{b}_{k-1} + \frac{1}{3}\mathbf{b}_k, \quad \mathbf{q}_k = \frac{1}{3}\mathbf{b}_{k-1} + \frac{2}{3}\mathbf{b}_k, \tag{3}$$

and

$$\mathbf{s}_k = \frac{\mathbf{q}_k + \mathbf{p}_k}{2}. \tag{4}$$

On the other hand, from [9] (Theorem 3.1), the points $\mathbf{b}_k, \mathbf{s}_k$, $k = 0, \ldots, n$, are now in $\mathbb{R}^3$ with $\mathbf{b}_0 = \mathbf{s}_0$ and $\mathbf{b}_n = \mathbf{s}_n$, and for $k = 1, \ldots, n-1$, we have

$$\mathbf{b}_k = \frac{\beta_{n-1-k}}{\beta_{n-1}}\Big[(-1)^k\mathbf{s}_0 + 6\sum_{j=1}^{k-1}(-1)^{k-j}\beta_{j-1}\mathbf{s}_j\Big] + \frac{\beta_{k-1}}{\beta_{n-1}}\Big[6\sum_{j=k}^{n-1}(-1)^{j-k}\beta_{n-1-j}\mathbf{s}_j + (-1)^{n-k}\mathbf{s}_n\Big], \tag{5}$$

where $\beta_k$ ($k = 0, \ldots, n-1$) is evaluated by the formula

$$\beta_k = 2^k\sum_{m=0}^{\lfloor k/2\rfloor}\binom{k+1}{k-2m}(3/4)^m. \tag{6}$$

Finally, we can give a simple process to obtain a so-called *relaxed, uniform B-spline space curve* $\mathscr{C}$ that interpolates an ordered set of points $\mathbf{s}_0$, $\mathbf{s}_1$, …, $\mathbf{s}_n$ (see [10]). The parametrization $\mathbf{f}$ of $\mathscr{C}$ is a piecewise cubic function on $[0,n]$ whose components $\mathbf{f}_k$, $k = 1, \ldots, n$, derived from (2) and (3), can be now given by

$$\begin{aligned}\mathbf{f}_k(t) = {} & (k-t)^3\mathbf{s}_{k-1} + (t-k+1)(k-t)^2(2\mathbf{b}_{k-1} + \mathbf{b}_k) \\ & + (t-k+1)^2(k-t)(\mathbf{b}_{k-1} + 2\mathbf{b}_k) + (t-k+1)^3\mathbf{s}_k,\end{aligned} \tag{7}$$

where $t \in [k-1,k]$, and the $\mathbf{b}_k$ are obtained from (5) for $k = 1, \ldots, n-1$, and $\mathbf{b}_0 = \mathbf{s}_0$ and $\mathbf{b}_n = \mathbf{s}_n$. Since $\mathbf{f}_1''(0) = \mathbf{f}_n''(n) = \mathbf{0}$, the curvatures of $\mathscr{C}$ at $t = 0$ and $t = n$ are both zero and we call $\mathscr{C}$ a *relaxed Bézier-spline space curve*.

We are interested in the implementation of the above parametrization by a Maple procedure. We list here some Maple commands that will appear in our procedures. They are all very important and frequently used in graphic and computation programming: `args`, `nargs`, `op`, `nops`, `ERROR`, `RETURN`, `convert`, `evalf`, `diff`, `expand`, `floor`, `for`, `fsolve`, `map`, `max`, `min`, `piecewise`, `plot`, `plot3d`, `proc`, `seq`, `solve`, `unapply`, and `while`; in addition, `LinearAlgebra` and `plots` are the great packages containing many procedures for specific purposes. A declaration to create a function, e.g., `f`, such as `f:=x->F(x)` or `f:=unapply(F(x),x)` (sub-procedures in `F(x)` are evaluated first), where `F(x)` is an expression or a list of expressions in `x`, is a very useful and convenient tool. In addition, the conditional structures `if-then-else` and `if-then-elif-else` are indispensable in branch programming, whereas the type "`list`" is a flexible ordered arrangement of operands (or components, elements) inside the square brackets `[,]`. See [11,12] and Maple help pages in each session to know more details about meaning, syntax, and usage of these commands, structures, and types. The implementation of some specific task by calling a procedure

name together with appropriate arguments is usually said to be a *calling sequence*. As a convention, we choose type `list` for elements of $\mathbb{R}^2$ or $\mathbb{R}^3$.

Now, let us make a procedure to compute $\mathbf{f}(t)$ on $[0, n]$ from (5)–(7), with $\mathbf{b}_0 = \mathbf{s}_0$ and $\mathbf{b}_n = \mathbf{s}_n$. It takes $S = \{\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n\}$ as its input and gives $\mathbf{f}(t)$ as its output in the form of $[F(t), G(t), H(t)]$ such that $F, G, H$ are the piecewise functions in $t$ on $[0, n]$. This procedure is called `BScurve3d` and its full code is given in the following.

BScurve3d

```
BScurve3d:=proc(Lst::list(list(realcons)))
local n,S,b,B,f,F,G,H,j,k;
n:=nops(Lst)-1:
for k from 0 to n do
S[k]:=Lst[k+1]:
end do:
for j from 0 to n-1 do
 B[j]:=2^j*add(binomial(j+1,j-2*m)*(3/4)^m,m=0..floor(j/2)):
end do:
for k from 1 to n-1 do
 b[k]:=(B[n-1-k]/B[n-1])*((-1)^k*S[0]+6*add((-1)^(k-j)*B[j-1]*S[j],j=1..k-1))
   +(B[k-1]/B[n-1])*((-1)^(n-k)*S[n]+6*add((-1)^(j-k)*B[n-1-j]*S[j],j=k..n-1)):
end do:
b[0]:=S[0]:
b[n]:=S[n]:
for k from 1 to n do
 f[k]:=unapply(expand((k-t)^3*S[k-1]+(t-k+1)*(k-t)^2*(2*b[k-1]+b[k])
   +(t-k+1)^2*(k-t)*(b[k-1]+2*b[k])+(t-k+1)^3*S[k]),t):
end do:
F:=t->piecewise(-1<=t and t<1,f[1](t)[1],
   seq([(k-1)<=t and t<k,f[k](t)[1]][],k=2..n-1),(n-1)<=t and t<n+1,f[n](t)[1]):
G:=t->piecewise(-1<=t and t<1,f[1](t)[2],
   seq([(k-1)<=t and t<k,f[k](t)[2]][],k=2..n-1),(n-1)<=t and t<n+1,f[n](t)[2]):
H:=t->piecewise(-1<=t and t<1,f[1](t)[3],
   seq([(k-1)<=t and t<k,f[k](t)[3]][],k=2..n-1),(n-1)<=t and t<n+1,f[n](t)[3]):
RETURN(unapply([F(t),G(t),H(t)],t));
end proc:
```

To declare a finite sequence of indexed expressions, we can use the operator `[]` to extract the contents of a list. For example, the command `[1,2,3][]` results in $1, 2, 3$, and the declaration of the sequence

$$1 \le t \text{ and } t < 2,\ f_2(t),\ 2 \le t \text{ and } t < 3,\ f_3(t),\ \ldots,\ n-2 \le t \text{ and } t < n-1,\ f_{n-1}(t)$$

can be written as: `seq([k-1<=t and t<k,f[k](t)][],k=2..n-1)`. We have used this declaration in `BScurve3d`.

If we have an ordered set of $(n+1)$ distinct points in $\mathbb{R}^3$ that are declared in Maple as a list of lists

$$S := [[a_0, b_0, c_0], [a_1, b_1, c_1], \ldots, [a_n, b_n, c_n]],$$

then we can obtain the position vector of the Bézier-spline curve $\mathscr{C}$ that interpolates these points by calling `f:=BScurve3d(S)`. The plot of $\mathscr{C}$ can be made by the procedure `spacecurve` in the `plots` package with the command

```
plots[spacecurve](f(t),t=0..n,opts);
```

The '`opts`' means *plotting options*. To implement these steps, for instance, we set

$$L := [[-2, 5, 1], [0, -1, 0], [-3, 1, 2], [1, 2, 3], [-5, -2, 1], [2, -4, -5], [0, 1, 7], [-6, 3, 2]]$$

and $\mathbf{f} := \texttt{BScurve3d}(L)$. Then, the plot of the Bézier-spline curve $\mathscr{C}$ that interpolates $L$ is given by the calling sequence

$$\texttt{plots[spacecurve]}(\mathbf{f}(t), t=0..7, \texttt{opts});$$

Figure 1 shows the curve $\mathscr{C}$ and the points of $L$. We take one more example of interpolating an ordered set of points on a given space curve to see how well a Bézier-spline curve fits this curve. Let $\mathscr{L}_1$ be a curve whose parametrization is $\mathbf{r}(t) = (t\cos t, 0.8t\sin t, t\cos t - \sin t)^T$, $t \in [-2.4, 5]$, with the declaration

$$\mathbf{r} := \texttt{t->[t*cos(t),(0.8)*t*sin(t),t*cos(t)-sin(t)]:}$$

Consider a partition of $[-2.4, 5]$ by the points $t_0 := -2.4$, $t_1 := -1.2$, $t_2 := 0.0$, $t_3 := 1.4$, $t_4 := 2.7$, $t_5 := 3.8$, $t_6 := 5.0$, and set

$$L_1 := [\mathbf{r}(t_0), \mathbf{r}(t_1), \ldots, \mathbf{r}(t_6)]:$$

The curve $\mathscr{L}_1$ and its approximation by a Bézier-spline curve $\mathscr{C}_1$ are also given in Figure 1.

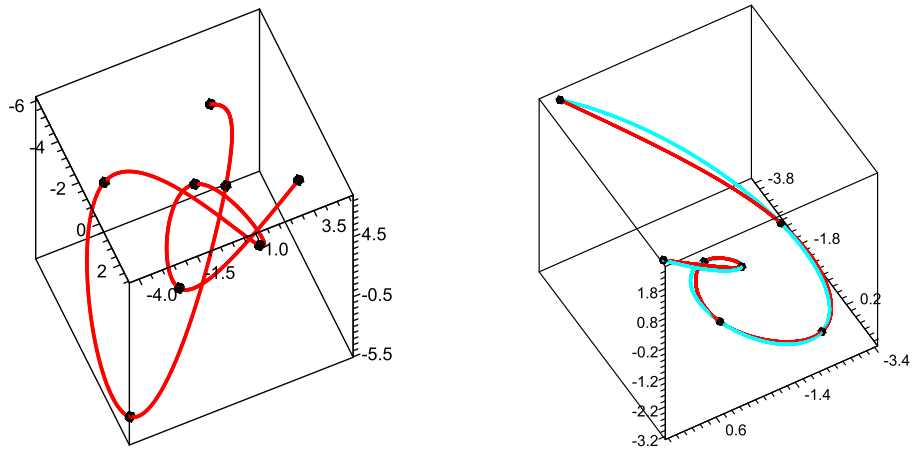


**Figure 1.** On the (**left**): The Bézier-spline curve $\mathscr{C}$ interpolates the list $L$. On the (**right**): The Bézier-spline curve $\mathscr{C}_1$ (in red) interpolates the data points $\mathbf{r}(t_0)$, $\mathbf{r}(t_1)$, ..., $\mathbf{r}(t_6)$ on the curve $\mathscr{L}_1$ (in cyan). `opts: axes=boxed, numpoints=2000, thickness=2, scaling=constrained`.

We will make some changes to obtain a so-called *closed, uniform B-spline space curve* $\mathscr{C}$ that interpolates an ordered set of points $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n$, with $\mathbf{s}_n = \mathbf{s}_0$ (see [10]). We call such a curve a *closed Bézier-spline space* curve. In this case, we choose appropriate settings to have again the relations (3) and (4) at the common point. The first setting should be $\mathbf{b}_n = \mathbf{b}_0$. Then, $\mathscr{C}$ is still composed of the cubic Bézier curves $\mathscr{C}_k$, $k = 1, \ldots, n$, as above. Specifically, at the interpolated point $\mathbf{s}_n = \mathbf{s}_0$, (4) becomes

$$\mathbf{s}_0 = \frac{\mathbf{p}_0 + \mathbf{q}_n}{2}, \tag{8}$$

where we have from (3) that

$$\mathbf{p}_0 = \frac{2}{3}\mathbf{b}_0 + \frac{1}{3}\mathbf{b}_1, \quad \mathbf{q}_n = \frac{1}{3}\mathbf{b}_{n-1} + \frac{2}{3}\mathbf{b}_n = \frac{1}{3}\mathbf{b}_{n-1} + \frac{2}{3}\mathbf{b}_0. \tag{9}$$

Now, from (8) and (9), we get the last setting

$$\mathbf{b}_0 = \frac{1}{4}(6\mathbf{s}_0 - \mathbf{b}_{n-1} - \mathbf{b}_1). \tag{10}$$

It is easy to have another Maple procedure, say `BScurve3dC`, for representing a closed Bézier-spline space curve $\mathscr{C}$ from the dataset $\{\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n\}$ with $\mathbf{s}_n = \mathbf{s}_0$, and the settings (10) and $\mathbf{b}_n = \mathbf{b}_0$. From the above discussion, the parametrization $\mathbf{f}$ of $\mathscr{C}$ has the components $\mathbf{f}_k$ given by (7), and we can check that

$$\mathbf{f}_1'(0) = \mathbf{f}_n'(n) = -3\mathbf{s}_0 + 2\mathbf{b}_0 + \mathbf{b}_1, \quad \mathbf{f}_1''(0) = \mathbf{f}_n''(n) = 6(\mathbf{s}_0 - \mathbf{b}_0).$$

Therefore, $\mathbf{f}$ is in $C^2([0, n])$ again and $\mathscr{C}_1, \mathscr{C}_n$ have the same curvature at their common point.

Let us take some examples on using `BScurve3dC`. As the steps to display closed Bézier-spline space curves are the same as for `BScurve3d`, we just give the graphical results of these examples. Note that the initial and terminal points of the input list for `BScurve3dC` have to be the same. Let $L$ be the list

$$L := [[0, 3, -1], [1, -2, 2], [-2, 0, -5], [2, 4, 3], [-1, 2, 4], [6, -1, 0], [0, 3, -1]].$$

The display of the closed Bézier-spline curve $\mathscr{C}_L$ that interpolates $L$ is given in Figure 2. Let $\mathbf{r}$ be the parametrization of a closed space curve $\mathscr{L}$ called a *trefoil knot* (see [1] (p. 897)) with

$$\mathbf{r}(t) := [(1 + 0.3\cos(3t))\cos(2t),\ (1 + 0.3\cos(3t))\sin(2t),\ 0.35\sin(3t)] \quad (t \in [0, 2\pi]).$$

The endpoints of $\mathscr{L}$ coincide with the point $[1.3, 0, 0]$. Let $T$ be a set of points on $\mathscr{L}$ such that

$$T := [[1.3, 0, 0], \mathbf{r}(0.5), \mathbf{r}(1.2), \mathbf{r}(1.8), \mathbf{r}(2.5), \mathbf{r}(3.2), \mathbf{r}(3.9), \mathbf{r}(4.5), \mathbf{r}(5.1), \mathbf{r}(5.8), [1.3, 0, 0]].$$

In Figure 2, we also give the display of $\mathscr{L}$ together with its approximation $\mathscr{C}_T$ that interpolates $T$.

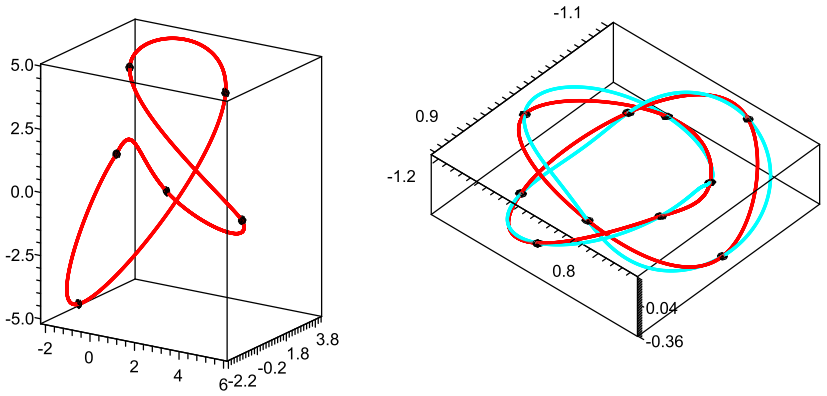


**Figure 2.** On the (**left**): The closed Bézier-spline curve $\mathscr{C}_L$ interpolates the list $L$. On the (**right**): The closed Bézier-spline curve $\mathscr{C}_T$ (in red) interpolates the set $T$ of points on the curve $\mathscr{L}$ (in cyan).

## 3. Computation of the Maximum Curvature of a Bézier-Spline Curve

Let $\mathscr{C}$ be a Bézier-spline curve that interpolates an ordered set $T$ of points $\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n$ in $\mathbb{R}^3$. Then $\mathbf{r}(t)$, the position vector of $\mathscr{C}$, is a piecewise cubic function of $C^2([0, n])$, given by its components $\mathbf{f}_k(t)$ in (7).

Avoiding the square root function, we have from (1):

$$K := \kappa^2 = \frac{\|\mathbf{v} \times \mathbf{a}\|^2}{v^6} = \frac{(\mathbf{v} \times \mathbf{a}) \cdot (\mathbf{v} \times \mathbf{a})}{(\mathbf{v} \cdot \mathbf{v})^3}.$$

Then, we have that

$$K' = \frac{[(\mathbf{v}\times\mathbf{a})\cdot(\mathbf{v}\times\mathbf{a})]'(\mathbf{v}\cdot\mathbf{v})^3 - [(\mathbf{v}\times\mathbf{a})\cdot(\mathbf{v}\times\mathbf{a})]3(\mathbf{v}\cdot\mathbf{v})^2 2(\mathbf{v}\cdot\mathbf{v}')}{(\mathbf{v}\cdot\mathbf{v})^6}$$
$$= \frac{2[(\mathbf{v}\times\mathbf{a})\cdot(\mathbf{v}\times\mathbf{a}')](\mathbf{v}\cdot\mathbf{v}) - 6[(\mathbf{v}\times\mathbf{a})\cdot(\mathbf{v}\times\mathbf{a})](\mathbf{v}\cdot\mathbf{a})}{(\mathbf{v}\cdot\mathbf{v})^4}.$$

As $\kappa$ attains its maximum value on $[0,n]$ only at solutions of the equation $K'=0$ or

$$(\mathbf{v}\cdot\mathbf{v})[(\mathbf{v}\times\mathbf{a})\cdot(\mathbf{v}\times\mathbf{a}')] - 3[(\mathbf{v}\times\mathbf{a})\cdot(\mathbf{v}\times\mathbf{a})](\mathbf{v}\cdot\mathbf{a}) = 0 \tag{11}$$

in the intervals $(i-1,i)$ and at their endpoints $i-1,i$, with $i=1,\ldots,n$, we can find $\kappa_{\max}$ on $[0,n]$ by the procedure `MaxCurvature3d`. However, at first, we present the procedure in the form of a pseudo-code algorithm. It would be easy to translate statements in such algorithms into Maple codes or other programming languages. Moreover, our discussion on how to use appropriate commands for a specific purpose will give a clear description of our procedures. In addition, we sometimes use built-in Maple procedures in those algorithms with their most simple form for convenience and simplicity.

Note that the left-hand side of (11) is a polynomial of degree at most 7. Letting $Q$ be such a polynomial (in one variable, say, $t$), we will use a powerful tool of Maple to find numerically all the zeros of $Q$ in a given interval. That is the procedure `fsolve` and it has been called in Algorithm 1 by the command: `fsolve(`$Q$`,`$t$`,`$\alpha..\beta$`)`. The output of this calling is a sequence of all real zeros of $Q$ in $[\alpha,\beta]$. Moreover, the expressions of dot and cross products are given by the great package of Maple: `LinearAlgebra`. We also select points in $[0,n]$ at which $\kappa$ attains its maximum value. Thus, the output of Algorithm 1 consists of $\kappa_{\max}$ and the set $\kappa^{-1}(\kappa_{\max})$ in $[0,n]$. Now, for relaxed Bézier-spline curves, we give the full code of `MaxCurvature3d` at the end of this section.

**Algorithm 1** Finding the maximum curvature of a Bézier-spline curve

**Input:** a set $T$ of $(n+1)$ points in $\mathbb{R}^3$;
**Output:** The maximum curvature $\kappa_{\max}$ of the Bézier-spline curve interpolating $T$;
1: $S_p := \{\}$;
2: **for** $i=1$ **to** $n$ **do**
3: $\mathbf{v} := \mathbf{f}_i'(t)$, $\mathbf{a} := \mathbf{f}_i''(t)$ // $\mathbf{f}_i$ from `BScurve3d`;
4: $Q :=$ the left-hand side of (11);
5: $P := \{$`fsolve(`$Q$`,`$t$`,`$i-1..i$`)`$\}$ // Solving (11) for $t\in[i-1,i]$;
6: $S_p := S_p \cup P$;
7: $S := \{F_i(t): t\in P\}\cup\{F_i(i-1), F_i(i)\}$ // $F_i := t\mapsto\kappa(t) = \|\mathbf{v}\times\mathbf{a}\|/v^3$, $v := \|\mathbf{v}\|$;
8: $m_i := \max S$;
9: **end for**
10: $\kappa_{\max} := \max\{m_1,\ldots,m_n\}$;
11: $M := S_p\cup\{0,1,\ldots,n\}$; $T_p := F^{-1}(\kappa_{\max})\subset M$ // $F := t\mapsto\kappa(t)$ on $[0,n]$, $F=F_i$ on $[i-1,i]$;
12: **return** $\kappa_{\max}$ and $T_p$;

`MaxCurvature3d`

```
MaxCurvature3d:=proc(L::list(list(realcons)))
local a,ad,A,AD,m,n,S,b,B,f,H,E,F,G,R,k,i,j,v,V,M,Kmax,N,P,Q,Sp,Tp,Tpoint;
n:=nops(L)-1:
for k from 0 to n do
S[k]:=L[k+1]:
end do:
for j from 0 to n-1 do
 B[j]:=2^j*add(binomial(j+1,j-2*m)*(3/4)^m,m=0..floor(j/2)):
end do:
```

```
for k from 1 to n-1 do
 b[k]:=(B[n-1-k]/B[n-1])*((-1)^k*S[0]+6*add((-1)^(k-j)*B[j-1]*S[j],j=1..k-1))
   +(B[k-1]/B[n-1])*((-1)^(n-k)*S[n]+6*add((-1)^(j-k)*B[n-1-j]*S[j],j=k..n-1)):
end do:
b[0]:=S[0]:
b[n]:=S[n]:
for k from 1 to n do
 f[k]:=unapply(expand((k-t)^3*S[k-1]+(t-k+1)*(k-t)^2*(2*b[k-1]+b[k])
   +(t-k+1)^2*(k-t)*(b[k-1]+2*b[k])+(t-k+1)^3*S[k]),t):
end do:
Sp:={}:
for i from 1 to n do
 v:=unapply(diff(f[i](t),t),t):
 a:=unapply(diff(f[i](t),t$2),t):
 ad:=unapply(diff(f[i](t),t$3),t):
 V:=convert(v(t),Vector):
 A:=convert(a(t),Vector):
 AD:=convert(ad(t),Vector):
 M:=expand(LinearAlgebra[DotProduct](V,V,conjugate=false)):
 N:=expand(LinearAlgebra[DotProduct](V,A,conjugate=false)):
 G:=map(expand,LinearAlgebra[CrossProduct](V,A)):
 H:=LinearAlgebra[CrossProduct](V,AD):
 E:=expand(LinearAlgebra[DotProduct](G,H,conjugate=false)):
 R:=expand(LinearAlgebra[DotProduct](G,G,conjugate=false)):
 Q:=expand(M*E-3*R*N):
 F[i]:=unapply(sqrt(abs(R))/abs(M)^(3/2),t):
 P:={fsolve(Q,t,i-1..i)}:
 m[i]:=max(seq(F[i](P[j]),j=1..nops(P)),F[i](i-1),F[i](i)):
 Sp:=Sp union P:
end do:
Kmax:=max(seq(m[j],j=1..n)):
Tpoint:={}:
for i from 1 to n do
 if (abs(F[i](i-1)-Kmax)=0) or (abs(F[i](i-1)-Kmax)=0.)  then
  Tpoint:= Tpoint union {i-1}:
 elif (abs(F[i](i)-Kmax)=0) or (abs(F[i](i)-Kmax)=0.)  then
  Tpoint:= Tpoint union {i}:
 end if:
end do:
if nops(Sp)=0 then
 RETURN(Kmax,Tpoint);
end if:
for j from 1 to nops(Sp) do
 if (floor(Sp[j])=n) then
  if (abs(F[n](Sp[j])-Kmax)=0 or abs(F[n](Sp[j])-Kmax)=0.)  then
   Tpoint:= Tpoint union {n}:
  end if:
 elif (abs(F[floor(Sp[j])+1](Sp[j])-Kmax)=0 or
            abs(F[floor(Sp[j])+1](Sp[j])-Kmax)=0.)  then
  Tpoint:= Tpoint union {Sp[j]}:
 end if:
end do:
RETURN(Kmax,Tpoint);
end proc:
```

Here we give an explanation of how to determine the set $\kappa^{-1}(\kappa_{\max}) = \{t \in [0,n] \colon \kappa(t) = \kappa_{\max}\}$. We first check whether $i$ ($\in \{0,1,\ldots,n\}$) belongs to this set, then we check the same for all solutions of the equation $Q = 0$. In addition, when we need the value $\mathbf{f}(t)$ for a point $t \in [0,n]$, we should take an appropriate component $\mathbf{f}_k$ of $\mathbf{f}$; if $t = n$, then we take $k = n$, else we take $k = \lfloor t \rfloor + 1$, since $\lfloor t \rfloor \le t < \lfloor t \rfloor + 1$.

We use `MaxCurvature3d` to determine $\kappa_{\max}$ in the examples whose graphical results are given in Figure 1. From the lists $L$ and $L_1$ in these examples, the calling sequences `MaxCurvature3d(`$L$`)` and `MaxCurvature3d(`$L_1$`)` result in

$$11.87724489,\ \{2.913967861\} \quad \text{and} \quad 3.135008280,\ \{2\},$$

respectively. If the parametrization of $\mathscr{C}_1$ is $\mathbf{h}$, then

$$\begin{aligned}\mathbf{f}(2.913967861) &= [1.06704678,\ 2.13642213,\ 2.94912425],\\ \mathbf{h}(2) &= [-0.8322936730,\ 1.454875883,\ -1.741591100]\end{aligned}$$

are the maximum curvature points on the curves $\mathscr{C}$ and $\mathscr{C}_1$, respectively.

The new version of `MaxCurvature3d` for closed Bézier-spline curves, say, `MaxCurvature3dC`, can be derived easily from Algorithm 1 with the modification "$\mathbf{f}_i$ from `BScurve3dC`". Then, we use `MaxCurvature3dC` to find $\kappa_{\max}$ of $\mathscr{C}_L$ displayed in Figure 2 and we obtain the result

$$1.652746390,\ \{1.979287163\}.$$

Let $\mathbf{g}$ be the parametrization of $\mathscr{C}_L$. The point

$$\mathbf{g}(1.979287163) = [-2.02583824,\ -0.094336298,\ -4.99371046]$$

is given in Figure 3 and this maximum point of curvature of $\mathscr{C}_L$ is very close to the interpolated point $[-2, 0, -5]$.

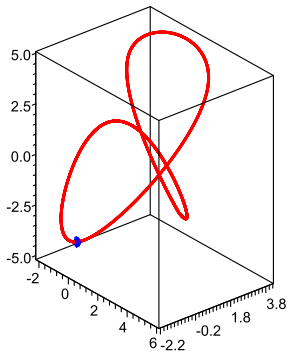


**Figure 3.** The value of $\kappa_{\max} = 1.652746390$ is attained at the blue point on $\mathscr{C}_L$.

To avoid a comparison error between fractions and decimal numbers, we may use the decimal point '.' for at least one component of the points in the `list` argument of `MaxCurvature3d`. Note that the result of `fsolve` only contains decimal numbers, so it will give us '2.', for instance, if it contains the integer '2'. To cover this case, we add a condition such as '$A - B = 0$.' in the definition of `MaxCurvature3d`, and it should be sometimes '$|A - B| < 10^{-m}$' with some positive integer $m$ when we need to obtain an expected result.

## 4. Remarks on the Two-Dimensional Case

For a relaxed Bézier-spline plane curve $\mathscr{L}$ that interpolates a dataset $M$ of points $\mathbf{s}_0$, $\mathbf{s}_1, \ldots, \mathbf{s}_n$ in $\mathbb{R}^2$, we have already a procedure to obtain its position vector $\mathbf{f}(t)$. That is just removing the lines `H:=t->`$\cdots$ and modifying the lines `RETURN(t->[F(t),G(t),H(t)])` to `RETURN(t->[F(t),G(t)])` in `BScurve3d`, and the remaining part is for `BScurve2d`, the Maple parametrization of $\mathscr{L}$. Maple provides the `plot` procedure to display plane curves with their parametrization $[x(t), y(t)]$, $t \in [a, b]$, by the declaration

`plot([`$x(t)$`,`$y(t)$`,`$t$`=`$a$`..`$b$`],opts);`

Accordingly, we first set $\mathbf{f} := \texttt{BScurve2d}(M)$, then we call

$$\texttt{plot([op(}\mathbf{f}(t)\texttt{),}t\texttt{=0..}n\texttt{],opts);} \quad \text{or} \quad \texttt{plot([}\mathbf{f}(t)\texttt{[],}t\texttt{=0..}n\texttt{],opts);}$$

to display $\mathscr{L}$. Similarly, we get `BScurve2dC`, the new version of `BScurve2d` for closed Bézier-spline plane curves, from `BScurve3dC`.

`MaxCurvature3d` can be modified to use only sets of points in $\mathbb{R}^2$ and we call its new version `MaxCurvature2d`. Let $\mathscr{L}$ be a smooth plane curve with parametrization $\mathbf{r}$. From the formula $\kappa(s) = \hat{\mathbf{T}}'(s) \cdot \hat{\mathbf{N}}(s)$ with arc length parameter $s$, we can write

$$\kappa = \frac{\hat{\mathbf{T}}'}{v} \cdot \hat{\mathbf{N}} = \frac{1}{v}\Big[\Big(\frac{1}{v}\Big)'\mathbf{v} + \frac{1}{v}\mathbf{a}\Big] \cdot \hat{\mathbf{N}} = \frac{1}{v^2}\mathbf{a} \cdot (J\hat{\mathbf{T}}) = \frac{\mathbf{a} \cdot \mathbf{u}}{v^3}$$

for a general parameter $t$, where

$$\mathbf{v} = \mathbf{r}', \quad v = \|\mathbf{v}\|, \quad \mathbf{u} = J\mathbf{v}, \quad \mathbf{a} = \mathbf{v}', \quad J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

We also consider $K := \kappa^2 = (\mathbf{a} \cdot \mathbf{u})^2/(\mathbf{v} \cdot \mathbf{v})^3$ and derive the following equation from $K' = 0$:

$$(\mathbf{v} \cdot \mathbf{v})(\mathbf{a} \cdot \mathbf{u})(\mathbf{a}' \cdot \mathbf{u}) - 3(\mathbf{a} \cdot \mathbf{u})^2(\mathbf{v} \cdot \mathbf{a}) = 0. \tag{12}$$

This equation has the same role as (11), so we can make a new version of `MaxCurvature3d`, say, `MaxCurvature2d`, following the steps in Algorithm 1 with some modification: there is no cross product in this version. The expression of the local variable `Q` in the definition of `MaxCurvature2d` is given by the left-hand side of (12). Note that the function $\mathbf{F}_i$ in the pseudo-code algorithm of `MaxCurvature2d` is now

$$\mathbf{F}_i := \frac{|\mathbf{a} \cdot \mathbf{u}|}{v^3}.$$

In the definition of `MaxCurvature2d`, the matrix $J$ is declared at right above `Sp:={}` by

```
J:=Matrix(2,2,[0,-1,1,0]):
```

and, then, we set `U:=J.V` inside the `for` loop at right below.

We give two more examples on getting the maximum curvature of a relaxed Bézier-spline plane curve and its maximum curvature points. Let $M$ be the list

$$M := [[0.5, 1], [0, 2], [0.8, 3], [1.7, 2.25], [2.5, 1.8], [3.5, 2.2], [3.2, 2.8]]$$

and $\mathscr{C}$ be the Bézier-spline curve that interpolates $M$. Then, we obtain the parametrization of $\mathscr{C}$ by setting

$$\mathbf{f} := \texttt{BScurve2d(}M\texttt{).}$$

On the other hand, the calling sequence `MaxCurvature2d(`$M$`)` gives us the result

$$5.011177204, \ \{5.168088495\}.$$

Thus, the maximum curvature of $\mathscr{C}$ is $\kappa_{\max} = 5.011177204$ and this value is attained at the point $\mathbf{f}(5.168088495) = [3.53273441, 2.298980995]$.

Let $\mathscr{L}$ be a curve with the parametrization $\mathbf{r}\colon\ t \mapsto [3\sin t,\ t\cos(3t)]$, $t \in [-1, 2]$, and let

$$t_0 := -1,\ t_1 := -0.6,\ t_2 := -0.2,\ t_3 := 0.2,\ t_4 := 0.6,\ t_5 := 0.9,\ t_6 := 1.3,\ t_7 := 1.7,\ t_8 := 2$$

be a partition of $[-1, 2]$. We set $N := [\mathbf{r}(t_0), \mathbf{r}(t_1), \mathbf{r}(t_2), \ldots, \mathbf{r}(t_8)]$. Letting $\mathscr{C}_1$ be the Bézier-spline curve that interpolates $N$, we get its parametrization

$$\mathbf{h} := \texttt{BScurve2d(}N\texttt{).}$$

Then, we derive

$$7.637332459,\ \{5.720743580\}$$

from the calling sequence `MaxCurvature2d(N)`. It follows that $\mathscr{C}_1$ attains the maximum curvature $\kappa_{\max} = 7.637332459$ at the point $\mathbf{h}(5.720743580) = [2.76963535, -1.07851249]$.

The results from the two examples above are given in Figure 4.

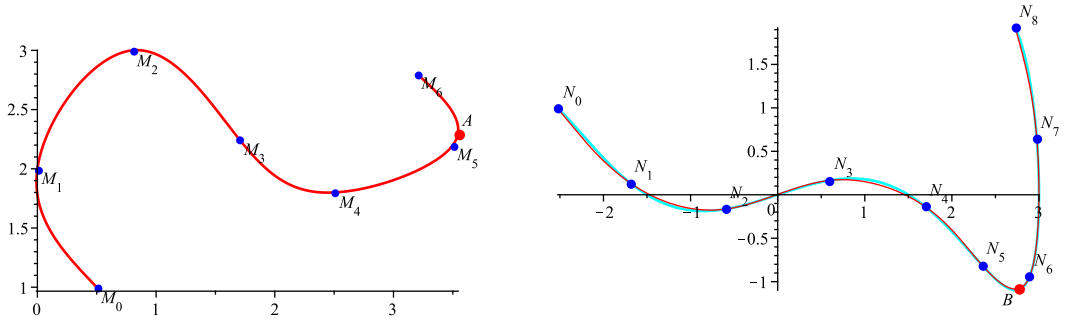


**Figure 4.** On the (**left**): The Bézier-spline curve $\mathscr{C}$ interpolates the dataset $M$. On the (**right**): The Bézier-spline curve $\mathscr{C}_1$ (in red) interpolates the set $N$ of points on $\mathscr{L}$ (in cyan). The red points $A$ and $B$ are the points of maximum curvature of $\mathscr{C}$ and $\mathscr{C}_1$, respectively.

The next examples are dealing with closed Bézier-spline curves. The datasets $A$ and $B$ in the first two examples are chosen to be symmetric to an axis, namely

$$A := [[3,1],[5,3],[4,4],[3,3],[2,4],[1,3],[3,1]];$$
$$B := [[1,4],[0.6,2],[2,0.4],[3.4,1],[2.6,2.8],[2.2,2.4],[4,1.6],[4.6,3],[3,4.4],[1,4]].$$

The curves $\mathscr{C}_A$ and $\mathscr{C}_B$ that interpolate $A$ and $B$, respectively, are given in Figure 5. The shapes of $\mathscr{C}_A$ and $\mathscr{C}_B$ can already be seen if we first display the interpolated points on the plane.

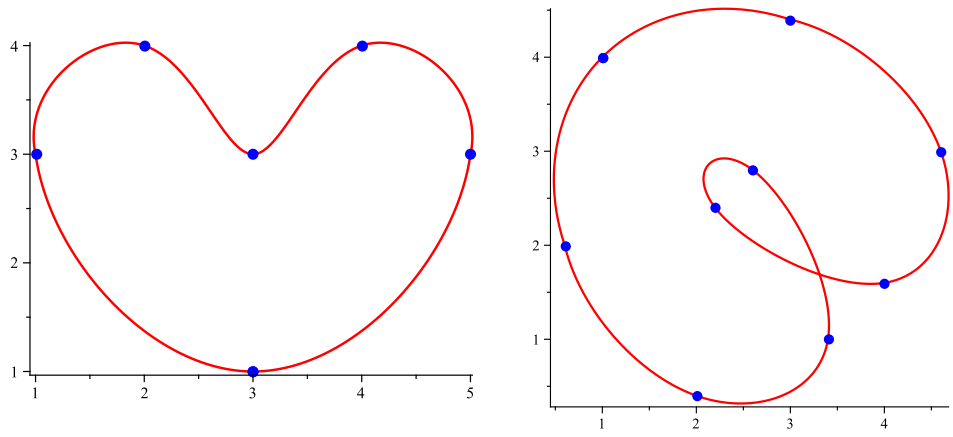

**Figure 5.** The Bézier-spline curves $\mathscr{C}_A$ (**left**) and $\mathscr{C}_B$ (**right**).

In the last two examples, we compute the maximum curvature of two closed Bézier-spline curves and show the maximum curvature points on these curves. The results are given in Figure 6. The interpolated points (on the right) in Figure 6 are chosen on the ellipse $x^2/9 + y^2/4 = 1$ (depicted in cyan).

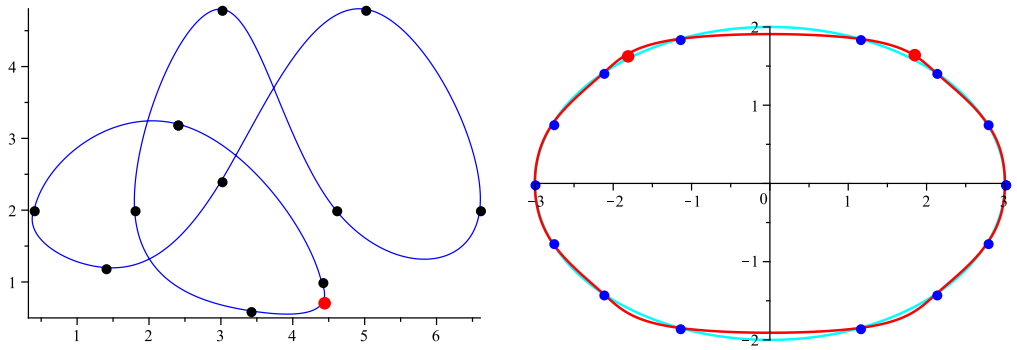

**Figure 6.** On the (**left**): $\kappa_{\max} = 5.045872521$. On the (**right**): $\kappa_{\max} = 0.9591183657$. These maximum curvatures are attained at the red points.

We sometimes want to compute the curvature of a Bézier-spline curve at a $p \in [0, n]$, so we should have a tool to do that. Algorithm 2 can be used to make such a tool.

**Algorithm 2** Finding the curvature of a Bézier-spline curve at its given point

**Input:** a set $T$ of $(n+1)$ points in $\mathbb{R}^2$, a point $p \in [0, n]$;
**Output:** The curvature $\kappa$ at $p$ of the Bézier-spline curve interpolating $T$;

1: **if** $p = n$ **then**
2: $\quad i := n$;
3: **else**
4: $\quad i := \lfloor p \rfloor + 1$;
5: **end if**
6: $\mathbf{v} := \mathbf{f}_i'(t)$, $\mathbf{a} := \mathbf{f}_i''(t)$ // $\mathbf{f}_i$ from `BScurve2d` or `BScurve2dC`;
7: $\mathbf{u} := J\mathbf{v}$ // $J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$;
8: $F := |\mathbf{a} \cdot \mathbf{u}|/v^3$ // $v := \|\mathbf{v}\|$, $F : t \mapsto \kappa(t)$;
9: **return** $F(p)$;

It is easy to derive the Maple procedure from Algorithm 2 that we call `Curvature2d` or `Curvature2dC`, depending on whether $\mathbf{f}_i$ is from `BScurve2d` or from `BScurve2dC`. Similarly, the extension of this procedure for the three-dimensional case can be obtained with the curvature function from Algorithm 1.

## 5. Conclusions

Our procedures may also be convenient tools for non-Maple users to estimate the maximum curvature of parametric smooth curves and to approximate a curve from its interpolated points by a Bézier-spline curve, as similarly done in [13–16]. They could be used for doing calculations to establish conditions for non-singularity of tubular surfaces and offset curves associated to a Bézier-spline curve, according to the issues profoundly presented in [17,18]. Moreover, our parametrization derived from the closed-form solution to a linear system for the interpolation problem would be better than those derived from the existing direct or iterative methods to solve the system, as noted in [19,20].

**Author Contributions:** H.P. constructed the parametrization of Bézier-spline curves and the algorithms. L.P.Q. wrote the Maple procedures. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Acknowledgments:** The authors want to thank the referees for their careful reading and their valuable corrections and suggestions. They also want to thank the Maplesoft experts for their great work in developing Maple, a powerful and user-friendly product.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References


1. Adams, R.A.; Essex, C. *Calculus: A Complete Course*, 9th ed.; Pearson Canada Inc.: Ontario, ON, Canada, 2018.
2. Montiel, S.; Ros, A. Curves and Surfaces, 2th ed. In *Graduate Studies in Mathematics*; American Mathematical Society: Rhode Island, RI, USA, 2009; Volume 69.
3. Yan, Z.; Schiller, S.; Wilensky, G.; Carr, N.; Schaefer, S. $\kappa$-Curves: Interpolation at local maximum curvature. *ACM Trans. Graph.* **2017**, *36*, 129. [CrossRef]
4. Miura, K.T.; Gobithaasan, R.U.; Salvi, P.;Wang, D.; Sekine, T.; Usuki, S.; Inoguchi, J.; Kajiwara, K. $\varepsilon\kappa$-Curves: Controlled local curvature extrema. *Vis. Comput.* **2022**, *38*, 2723—2738. [CrossRef]
5. Ahn, Y.J.; Kim, H.O. Curvatures of the Quadratic Rational Bézier Curves. *Comput. Math. Appl.* **1998**, *36*, *9*, 71–83. [CrossRef]
6. Cantón, A.; Fernández-Jambrina, L.; Vázquez-Gallo, M.J. Curvature of planar aesthetic curves. *J. Comput. Appl. Math.* **1998**, *381*, 113042. [CrossRef]
7. Wang, A; Zhao, G.; Hou, F. Constructing Bézier curves with monotone curvature. *J. Comput. Appl. Math.* **2019**, *355*, 1–10. [CrossRef]
8. He, C.; Zhao, G.; Wang, A.; Li, S.; Cai, Z. Planar Typical Bézier Curves with a Single Curvature Extremum. *Mathematics* **2021**, *9*, 2148. [CrossRef]
9. Quan, L.P.; Nhan, T.A. A Closed-Form Solution to the Inverse Problem in Interpolation by a Bézier-Spline Curve. *Arab. J. Math.* **2020**, *9*, 155–165. [CrossRef]
10. Michael, S. Cubic B-Splines Using PSTricks (A Guiding Document for the Package `Pst-Bspline` (Version 1.62 2016-04-21)). Available online: https://ctan.org/pkg/pst-bspline (accessed on 30 January 2023).
11. Monagan, M.B.; Geddes, K.O.; Heal, K.M.; Labahn, G.; Vorkoetter, S.M.; McCarron, J.; DeMarco, P. *MAPLE Introductory Programming Guide*; Waterloo MAPLE Inc.: Ontario, ON, Canada, 2008.
12. Monagan, M.B.; Geddes, K.O.; Heal, K.M.; Labahn, G.; Vorkoetter, S.M.; McCarron, J.; DeMarco, P. *MAPLE Advanced Programming Guide*; Waterloo MAPLE Inc.: Ontario, ON, Canada, 2008.
13. Chen, X.D.; Ma, W.; Paul, J.C. Cubic B-spline curve approximation by curve unclamping. *Comput.-Aided Des.* **2010**, *42*, 523–534. [CrossRef]
14. Cheng, F.; Barsky, B.A. Interproximation: Interpolation and approximation using cubic spline curves. *Comput.-Aided Des.* **1991**, *23*, 700–706. [CrossRef]
15. Hoschek, J. Spline approximation of offset curves. *Comput. Aided Geom. Des.* **1988**, *5*, 33–40. [CrossRef]
16. Kozak, J.; Krajnc, M. Geometric interpolation by planar cubic polynomial curves. *Comput. Aided Geom. Des.* **2007**, *24*, 67-–78. [CrossRef]
17. Maekawa, T.; Patrikalakis, N.M. Computation of singularities and intersections of offsets of planar curves. *Comput. Aided Geom. Des.* **1990**, *7*, 101–127. [CrossRef]
18. Maekawa, T.; Patrikalakis, N.M.; Sakkalis, T.; Yu, G. Analysis and applications of pipe surfaces. *Comput. Aided Geom. Des.* **1998**, *15*, 437–458. [CrossRef]
19. Farin, G. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*, 5th ed.; Morgan Kaufmann: California, CA, USA, 2001.
20. Hartmut, P.; Wolfgang B.; Marco, P. *Bézier and B-Spline Techniques*; Springer: Berlin/Heidelberg, Germany; New York, NY, USA, 2002.