Int. j. adv. multidisc. res. stud. 2024; 4(4):738-753

Received: 18-06-2024
Accepted: 28-07-2024

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

Applied for Higher Order Bezier Curve using MATLAB

[1] Taha Gabaireldar Elradi, [2] Subhi Abdalazim Aljily Osman, [3] Musa Adam Abdullah

[1] Department of Mathematics, Faculty of Science, University of Albutana, Sudan
[2] Department of Mathematics, Faculty of Computer Science and Information Technology, University of Albutana, Sudan
[3] Department of Mathematics, Faculty of Computer Science and Information Technology, University of the Holy Quran and Tassel of Science, Sudan

DOI: https://doi.org/10.62225/2583049X.2024.4.4.3102
Corresponding Author: **Taha Gabaireldar Elradi**

Abstract

A Bézier curve is a parametric curve used in computer graphics and related fields. A set of discrete "control points" defines a smooth, continuous curve by means of a formula. The piecewise nature of a Bezier curve means that its representative equation is a linear combination of Bezier of particular degrees. The aim of this study is to infer Bézier curve graphs from higher orders using MATLAB. Where the applied analytical method was used I have reached .Will be clarified, and the following results have been reached t is used in the linear curve association to express the distance ($B(t)$ on the line between the points P_0 and P_1, In a second-order Bezier curve, we can create intermediate points such as Q_0 and Q_1. These points have a value located in the interval $[1,0]$, To obtain a Bezier curve for higher degrees, note that the idea in arriving at drawing the curve is to divide each line by placing a point on it and then connect the new points to each other (note that we get rid of a point at each stage), and we repeat the process until we have only one line left. So, we put ($B(t)$ on it and then we draw the curve. By comparing the manual solution and the solution using MATLAB, we find that the solution using MATLAB is faster and more accurate, especially at higher levels. And the study recommends the following : It is noted that the curve lies completely within the convex closure of the points, so these points are used with ease to form the required curve, it is also possible to apply geometric transformations (such as rotation and sliding) to the curve by applying this transformation to the control points of this curve.

Keywords: Bezier Curve, MATLAB, Pascal's Triangle, Sudan

Introduction

The curve theory is used in differential geometry, kinematics, robotics and engineering. Bezier curves are important subset of the curves. Bezier curves have been used in computer-aided geometric design (CAGD) and in many areas. For every Bezier curve, the control points are uniquely defined . In CAGD, the most popular areas of research are used shape control parameters to construct Bezier curves(Senay Baydas and Bluent Karakas, Defining a curve as a Bezier curve, Journal of Taibah University for Science, 2019).

A Bézier curve is a parametric curve that is used to draw the shapes in the fields of computer graphics and computer-aided geometric design. The Bézier curve is usually followed by the defining polygon. The tangent vectors direction at the end is the same as the vector defined by the first and last segment of the Bézier curves. It is useful in many industrial and engineering fields with various applications. Since Bézier curve always mimics the shape of its control polygon, designers can easily attain the required shape for designing purposes. (Salma Naseer and others, A Class of Sextic Trigonometric Bezier Curve with Two Shape Parameters, researchgate, 2021).

The construction of curves and surfaces is a key issue in computer aided geometric design (CAGD). The CAGD method arise from the need of the efficient computer representation of practical curves and surfaces, which is very broadly used in engineering design. In CAGD, the parametric curves is the combination of basis functions and control points. The parametric representation of curves and surfaces with shape parameters have received attention in recent years. In recent years, the trigonometric with shape parameters play a very important role in CAGD in the design of curves. Many works have been done with the help of trigonometric polynomial for the representation of the curves and surfaces. Therefore, it is clear that Bézier curves, the quadratic and cubic Bézier curves, have very wide applications. In recent years, trigonometric polynomial curves like those of Bézier type are considerably in discussion (Mridula Dube and Bharti Yadav, The Quintic Trigonometric Bézier

Curve with single Shape Parameter, International Journal of Scientific and Research Publications, 2014).
Bezier curves (BC) were independently developed by Casteljau and Bézicr in the last century, and while their origin can be traced to the design of car body shapes in the automotive industry, their use is no longer confined to this field. Indeed, their robustness in curve and surface representation means they pervade many areas of multimedia technology including shape description of characters and objects, active shape lip modelling, shape coding and error concealment for MPEG-4 coded objects and surface mapping.(Ferdous Sohel and others, A dynamic Bezier curve model, semanticscholar, 2005).
in this area obtaining new curves have been very important for CAD and CAM. The most known and commonly used in CAD/CAM curves are Bezier curves. (Aslı Ayar and Bayram Sahin, Curves used in highway design and Bezier curves, doi.org, 2022).
B´ ezier Curves were invented in 1962 by the French engineer Pierre B´ ezier for designing automobile bodies. Today B´ ezier Curves are widely used in computer graphics and animation. The B´ ezier curves have useful properties for the path generation problem. (Jiwoong Choi and others, Smooth Path Generation Based on B´ezier Curves for Autonomous Vehicles, iaeng.org, 2009).

1. The B´ezier Curve:

The B´ezier curve is a parametric curve P(t) that is a polynomial function of the parameter t. The degree of the polynomial depends on the number of points used to define the curve. The method employs control points and produces an approximating curve (note the title of this paper). The curve does not pass through the interior points but is attracted by them. It is as if the points exert a pull on the curve. Each point influences the direction of the curve by pulling it toward itself, and that influence is strongest when the curve gets nearest the point. Fig 1 shows some examples of cubic B´ezier curves. Such a curve is defined by four points and is a cubic polynomial. Notice that one has a cusp and another one has a loop. The fact that the curve does not pass through the points implies that the points are not “set in stone” and can be moved. This makes it easy to edit, modify and reshape the curve, which is one reason for its popularity. The curve can also be edited by adding new points, or deleting points. But they are cumbersome because the mathematical expression of the curve depends on the number of points, not just on the points themselves.

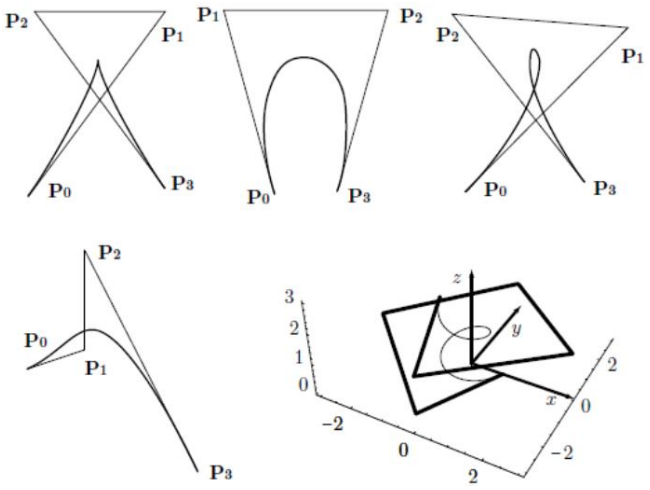

Fig 1: Four Plane Cubic and One Space B´ezier Curves with their Control Points and Polygons

The control polygon of the B´ezier curve is the polygon obtained when the control points are connected, in their natural order, with straight segments. How does one go about deriving such a curve? We describe two approaches to the design a weighted sum and a linear interpolation and show that they are identical.

1.1 Pascal Triangle and the Binomial Theorem

The Pascal triangle and the binomial theorem are related because both employ the same numbers. The Pascal triangle is an infinite triangular matrix that’s built from the edges inside

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
… … …

We first fill the left and right edges with ones, then compute each interior element as the sum of the two elements directly above it. As can be expected, it is not hard to obtain an explicit expression for the general element of the Pascal triangle. We

first number the rows from 0 starting at the top, and the columns from 0 starting on the left. A general element is denoted by $\binom{i}{j}$. We then observe that the top two rows (corresponding to $i = 0,1$) consist of 1's and that every other row can be obtained as the sum of its predecessor and a shifted version of its predecessor. For example,

$$\begin{array}{r} 1\ \ 3\ \ 3\ \ 1 \\ +\ 1\ \ 3\ \ 3\ \ 1 \\ \hline 1\ \ 4\ \ 6\ \ 4\ \ 1 \end{array}$$

This shows that the elements of the triangle satisfy

$$\binom{i}{0} = \binom{i}{i} = 1, \qquad i = 0,1, \ldots$$

$$\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}, \qquad i = 2,3, \ldots, \quad j = 1,2, \ldots, (i-1)$$

From this it is easy to derive the explicit expression

$$\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$$

$$= \frac{(i-1)!}{(j-1)!\,(i-j)!} + \frac{(i-1)!}{j!\,(i-1-j)!}$$

$$= \frac{j(i-1)!}{j!\,(i-j)!} + \frac{(i-j)(i-1)!}{j!\,(i-j)!}$$

$$= \frac{i!}{j!\,(i-j)!}$$

Thus, the general element of the Pascal triangle is the well-known binomial coefficient

$$\binom{i}{j} = \frac{i!}{j!\,(i-j)!}$$

The binomial coefficient is one of Newton's many contributions to mathematics.
His binomial theorem states that

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i} \tag{1}$$

This equation can be written in a symmetric way by denoting $j = n - i$. The result is

$$(a+b)^n = \sum_{i,j>0}^{i+j=n} \frac{(i+j)!}{i!\,j!} a^i b^j \tag{2}$$

From which we can easily guess the trinomial theorem

$$(a+b+c)^n = \sum_{i,j,k>0}^{i+j+k=n} \frac{(i+j+k)!}{i!\,j!\,k!} a^i b^j c^k \tag{3}$$

1.2 The Bernstein form of the B´ezier Curve

The first approach to the B´ezier curve expresses it as a weighted sum of the points (with, of course, barycentric weights). Each control point is multiplied by a weight and the products are added. We denote the control points by $P_0, P_1, \ldots, P_n$ (n is therefore defined as 1 less than the number of points) and the weights by Bi. The expression of weighted sum is

$$P(t) = \sum_{i=0}^{n} p_i B_{i,n}(t) \quad 0 \le t \le 1$$

The result, $P(t)$, depends on the parameter t. Since the points are given by the user, they are fixed, so it is the weights that must depend on t. We therefore denote them by $B_i(t)$. How should $B_i(t)$ behave as a function of t?

We first examine $B_0(t)$, the weight associated with the first point P_0. We want that point to affect the curve mostly at the beginning, i.e., when t is close to 0. Thus, as t grows toward 1 (i.e., as the curve moves away from P_0), $B_0(t)$ should drop down to 0. When $B_0(t) = 0$, the first point no longer influences the shape of the curve.

Next, we turn to $B_1(t)$. This weight function should start small, should have a maximum when the curve approaches the second point P_1, and should then start dropping until it reaches zero. A natural question is: When (for what value of t) does the curve reach its closest approach to the second point? The answer is: It depends on the number of points. For three points (the case $n = 2$), the B´ezier curve passes closest to the second point (the interior point) when $t = 0.5$. For four points, the curve is nearest the second point when $t = \frac{1}{3}$. It is now clear that the weight functions must also depend on n and we denote them by $B_{n,i}(t)$. Hence, $B_{3,1}(t)$ should start at 0, have a maximum at $t = \frac{1}{3}$, and go down to 0 from there. Fig 2 shows the desired behavior of $B_{n,i}(t)$ for n = 2, 3, and 4. The five different weights $B_{4,i}(t)$ have their maxima at $t = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$, and 1.

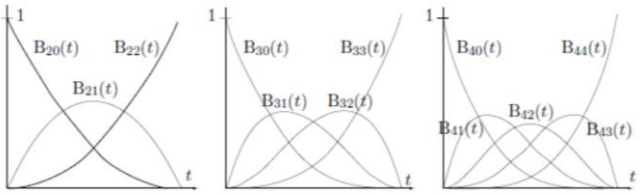

Fig 2: The Bernstein Polynomials for n = 2, 3, 4

The functions chosen by B´ezier (and also by de Casteljau) were derived by the Russian mathematician Sergeĭ Natanovich Bernshteĭn in 1912. They are known as the Bernstein polynomials and are defined by

$$B_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \text{where } \binom{n}{i} = \frac{n!}{i!\,(n-i)!} \tag{4}$$

are the binomial coefficients. These polynomials feature the desired behavior and have a few more useful properties that are discussed here. (In calculating the curve, we assume that the quantity 0^0, which is normally undefined, equals 1.)

The B´ezier curve is now defined as

$$P(t) = \sum_{i=0}^{n} p_i B_{i,n}(t) \text{ where } B_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i} \text{ and } 0 \le t \le 1 \tag{5}$$

Each control point (a pair or a triplet of coordinates) is multiplied by its weight, which is in the range $[0, 1]$. The weights act as blending functions that blend the contributions of the different points.

When B´ezier started searching for such functions in the early 1960s, he set the following requirements:

1. The functions should be such that the curve passes through the first and last control points.
2. The tangent to the curve at the start point should be $P_1 - P_0$, i.e., the curve should start at point P_0 moving toward P_1. A similar property should hold at the last point.
3. The same requirement is generalized for higher derivatives of the curve at the two extreme endpoints. Hence, $P^{tt}(0)$ should depend only on the first point P_0 and its two neighbors P_1 and P_2. In general, $P^{(k)}(0)$ should only depend on P_0 and its k neighbors P_1 through P_k. This feature provides complete control over the continuity at the joints between separate B´ezier curve segments.
4. The weight functions should be symmetric with respect to t and $(1 - t)$. This means that a reversal of the sequence of control points would not affect the shape of the curve.
5. The weights should be barycentric, to guarantee that the shape of the curve is independent of the coordinate system.
6. The entire curve lies within the convex hull of the set of control points.

The definition shown in Equation (5), using Bernstein polynomials as the weights, satisfies all these requirements. In particular, requirement 5 is proved when Equation (1) is written in the form $[t+(1-t)]n = \cdots$ (see Equation (10) if you cannot figure this out). Following are the explicit expressions of these polynomials for $n = 2, 3, \text{and } 4$.

Example (1): For $n = 2$ (three control points), the weights are

$$B_{2,0}(t) = \binom{2}{0} t^0 (1-t)^{2-0} = (1-t)^2,$$
$$B_{2,1}(t) = \binom{2}{1} t^1 (1-t)^{2-1} = 2t(1-t),$$
$$B_{2,2}(t) = \binom{2}{2} t^2 (1-t)^{2-2} = t^2,$$

and the curve is

$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$
$$= ((1-t)^2, 2t(1-t), t^2)(P_0, P_1, P_2)^T \quad (6)$$
$$= (t^2, t, 1)\begin{pmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} P_0 \\ P_1 \\ P_2 \end{pmatrix}$$

MATLAB code to compute the quadratic Bézier curve for n=2 with three control points P_0, P_1, and P_2.

```
% Define the control points (replace these with actual values)
P0 = [0, 0]; % Example values, replace with your actual coordinates
P1 = [1, 2]; % Example values, replace with your actual coordinates
P2 = [2, 0]; % Example values, replace with your actual coordinates
% Define the parameter t from 0 to 1
t = linspace(0, 1, 100);
% Calculate the Bernstein polynomials
B_2_0 = (1 - t).^2; B_2_1 = 2 * t .* (1 - t); B_2_2 = t.^2;
% Calculate the curve points
P = B_2_0' * P0 + B_2_1' * P1 + B_2_2' * P2;
% Plot the quadratic Bézier curve
figure; plot(P(:,1), P(:,2), 'b', 'LineWidth', 2); hold on;
plot([P0(1) P1(1) P2(1)], [P0(2) P1(2) P2(2)], 'ro--'); % Plot control
points and polygon
legend('Bézier Curve', 'Control Polygon'); title('Quadratic Bézier Curve');
xlabel('x'); ylabel('y'); grid on; axis equal;
```

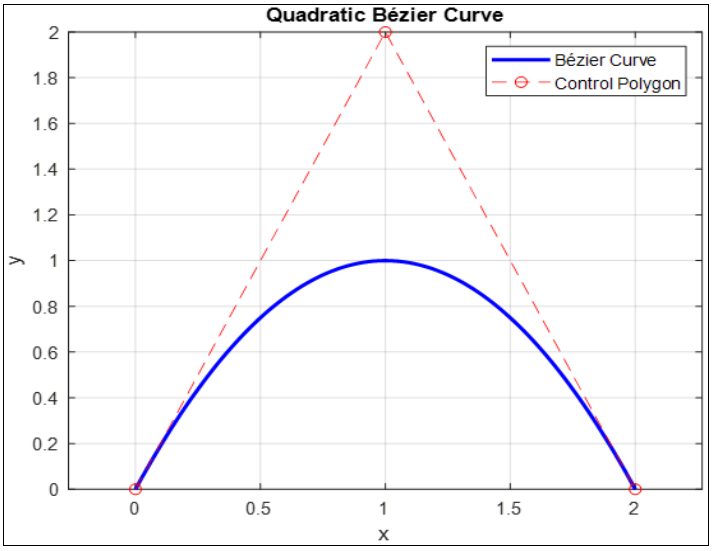

Fig 3: Bézier curve for n=2 with three control points P_0, P_1, and P_2

In the special case $n = 3$, the four weight functions are

$$B_{3,0}(t) = \binom{3}{0} t^0 (1-t)^{3-0} = (1-t)^3,$$

$$B_{3,1}(t) = \binom{3}{1} t^1 (1-t)^{3-1} = 3t(1-t)^2,$$

$$B_{3,2}(t) = \binom{3}{2} t^2 (1-t)^{3-2} = 3t^2(1-t),$$

$$B_{3,3}(t) = \binom{3}{3} t^3 (1-t)^{3-3} = t^3,$$

And the curve is

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \quad (7)$$

$$= [(1-t)^3, 3t(1-t)^2, 3t^2(1-t), t^3][P_0, P_1, P_2, P_3]^T$$

$$= [(1-3t+3t^2-t^3), (3t-6t^2+3t^3), (3t^2-3t^3), t^3][P_0, P_1, P_2, P_3]^T$$

$$= (t^3, t^2, t, 1) \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} \quad (8)$$

It is clear that $P(t)$ is a cubic polynomial in t. It is the cubic B´ezier curve.

The B´ezier curve can also be represented in a very compact and elegant way as $P(t) = (1 - t + tE)^n P_0$, where E is the shift operator defined by $EP_i = P_{i+1}$ (i.e., applying E to point P_i produces point P_{i+1}). The definition of E implies $EP_0 = P_1$, $E^2P_0 = P_2$, and $E^iP_0 = P_i$. The B´ezier curve can now be written

$$P(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} p_i = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} E^i p_0$$

$$= \sum_{i=0}^{n} \binom{n}{i} (tE)^i (1-t)^{n-i} p_0$$

$$= (tE + (1-t))^n p_0$$

Where the last step is an application of the binomial theorem, Equation (*1*).

MATLAB code to compute the cubic Bézier curve for $n=3$ with four control points P_0, P_1, P_2, and P_3.

```
% Define the control points (replace these with actual values)
P0 = [0, 0]; % Example values, replace with your actual coordinates
P1 = [1, 3]; % Example values, replace with your actual coordinates
P2 = [3, 3]; % Example values, replace with your actual coordinates
P3 = [4, 0]; % Example values, replace with your actual coordinates
% Define the parameter t from 0 to 1
t = linspace(0, 1, 100);
% Calculate the Bernstein polynomials
B_3_0 = (1 - t).^3; B_3_1 = 3 * t .* (1 - t).^2;
B_3_2 = 3 * t.^2 .* (1 - t); B_3_3 = t.^3;
% Calculate the curve points
P = B_3_0' * P0 + B_3_1' * P1 + B_3_2' * P2 + B_3_3' * P3;
% Plot the cubic Bézier curve
figure; plot(P(:,1), P(:,2), 'b', 'LineWidth', 2); hold on;
plot([P0(1) P1(1) P2(1) P3(1)], [P0(2) P1(2) P2(2) P3(2)], 'ro--'); % Plot
control points and polygon
legend('Bézier Curve', 'Control Polygon'); title('Cubic Bézier Curve');
xlabel('x'); ylabel('y'); grid on; axis equal;
```

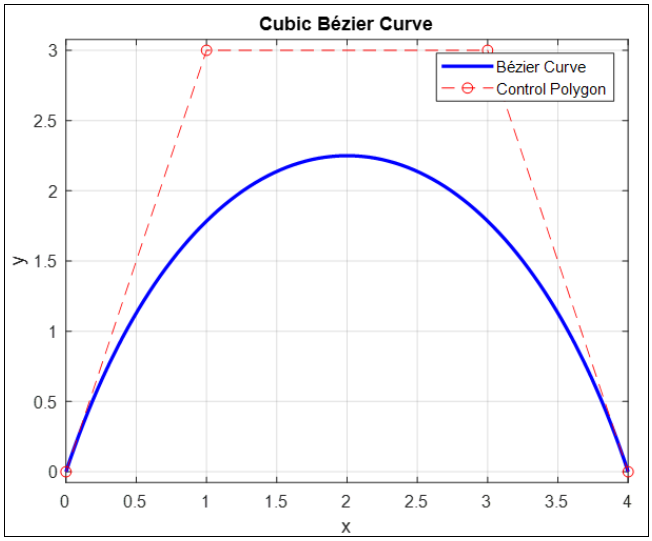

Fig 4: Bézier curve for n=3 with four control points $P_0, P_1, P_2,$ and P_3

Example (2): For $n = 1$, this representation amounts to

$$P(t) = (1 - t + tE)P_0 = P_0(1 - t) + P_1 t.$$

For $n = 2$, we get

$$P(t) = (1 - t + tE)^2 P_0$$

$$= (1 - t + tE - t + t^2 - t^2E + tE - t^2E + t^2E^2)P_0$$

$$= P_0(1 - 2t + t^2) + P_1(2t - 2t^2) + P_2 t^2$$

$$= P_0(1 - t)^2 + P_1 2t(1 - t) + P_2 t^2$$

Given $n + 1$ control points P_0 through P_n, we can represent the B´ezier curve for the points by $P_n^{(n)}(t)$, where the quantity $P_i^{(j)}(t)$ is defined recursively by

$$P_i^{(j)}(t) = \begin{cases} (1-t)P_{i-1}^{(j-1)}(t) + tP_i^{(j-1)}(t), & \text{for } j > 0 \\ P_i, & \text{for } j = 0 \end{cases} \quad (9)$$

The following examples show how the definition above is used to generate the quantities $P_i^{(j)}(t)$ and why $P_n^{(n)}(t)$ is the degree-n curve:

$$P_0^{(0)}(t) = P_0\ ,\quad P_1^{(0)}(t) = P_1\ , P_2^{(0)}(t) = P_2\ ,\ \ldots\ , P_n^{(0)}(t) = P_n\ ,$$

$$P_1^{(1)}(t) = (1-t)P_0^{(0)}(t) + tP_1^{(0)}(t) = (1-t)P_0 + tP_1\ ,$$

$$P_2^{(2)}(t) = (1-t)P_1^{(1)}(t) + tP_2^{(1)}(t)$$

$$= (1-t)\big((1-t)P_0 + tP_1\big) + t\big((1-t)P_1 + tP_2\big)$$

$$= (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2\ ,$$

$$P_3^{(3)}(t) = (1-t)P_2^{(2)}(t) + tP_3^{(2)}(t)$$

$$= (1-t)\left((1-t)P_1^{(1)}(t) + tP_2^{(1)}(t)\right) + t\left((1-t)P_2^{(1)}(t) + tP_3^{(1)}(t)\right)$$

$$= (1-t)^2 P_1^{(1)}(t) + 2t(1-t)P_2^{(1)}(t) + t^2 P_3^{(1)}(t)$$

$$= (1-t)^2\big((1-t)P_0 + tP_1\big) + 2t(1-t)\big((1-t)P_1 + tP_2\big) + t^2((1-t)P_2 + tP_3$$

$$= (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 .$$

MATLAB code to generate Bézier curves for n=1 using the recursive definition provided.

```
% Define the control points (replace these with actual values)
P0 = [0, 0]; % Example values, replace with your actual coordinates
P1 = [1, 1]; % Example values, replace with your actual coordinates
% Define the parameter t from 0 to 1
t = linspace(0, 1, 100);
% Compute the Bézier curve for n = 1
P = (1 - t)' * P0 + t' * P1;
% Plot the linear Bézier curve
figure; plot(P(:,1), P(:,2), 'b', 'LineWidth', 2); hold on;
plot([P0(1) P1(1)], [P0(2) P1(2)], 'ro--'); % Plot control points and
polygon
legend('Bézier Curve', 'Control Polygon');
title('Linear Bézier Curve (n = 1)'); xlabel('x'); ylabel('y');
grid on; axis equal;
```

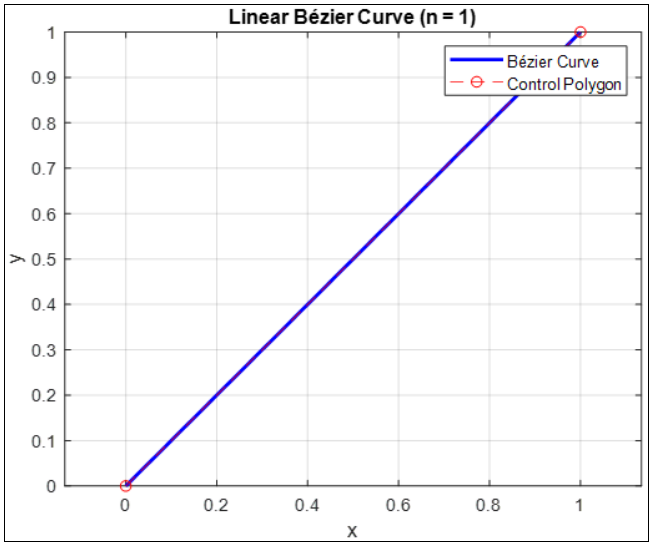

Fig 5: Bézier curve for n = 1 using the recursive definition

MATLAB code to generate Bézier curve for n = 2 using the recursive definition

```
% Define the control points (replace these with actual values)
P0 = [0, 0]; % Example values, replace with your actual coordinates
P1 = [1, 2]; % Example values, replace with your actual coordinates
P2 = [2, 0]; % Example values, replace with your actual coordinates
% Define the parameter t from 0 to 1
t = linspace(0, 1, 100);
% Compute the Bézier curve for n = 2 using the recursive definition
P0_0 = repmat(P0, length(t), 1); P1_0 = repmat(P1, length(t), 1);
P2_0 = repmat(P2, length(t), 1); P1_1 = (1 - t).' .* P0_0 + t.' .* P1_0;
P2_1 = (1 - t).' .* P1_0 + t.' .* P2_0; P2_2 = (1 - t).' .* P1_1 + t.' .*
P2_1;
% Plot the quadratic Bézier curve
figure; plot(P2_2(:,1), P2_2(:,2), 'b', 'LineWidth', 2); hold on;
plot([P0(1) P1(1) P2(1)], [P0(2) P1(2) P2(2)], 'ro--'); % Plot control
points and polygon
legend('Bézier Curve', 'Control Polygon');
title('Quadratic Bézier Curve (n = 2)'); xlabel('x'); ylabel('y');
grid on; axis equal;
```

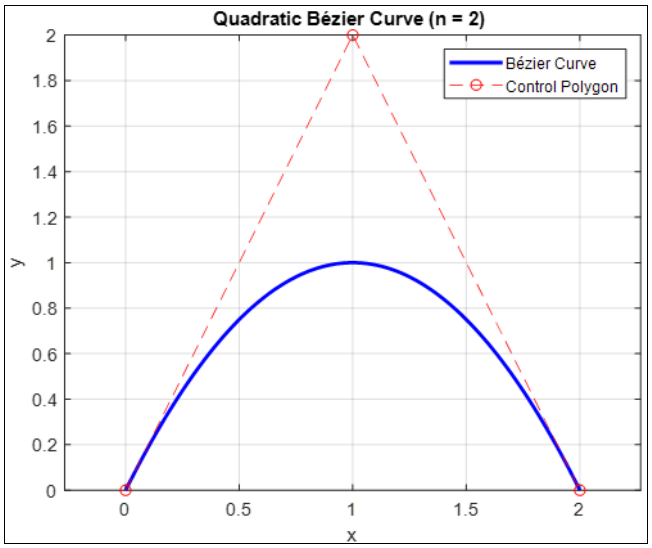

Fig 6: Bézier curve for n = 2 using the recursive definition

MATLAB code to generate Bézier curve for n = 3 using the recursive definition

```
% Define the control points (replace these with actual values)
P0 = [0, 0]; % Example values, replace with your actual coordinates
P1 = [1, 3]; % Example values, replace with your actual coordinates
P2 = [3, 3]; % Example values, replace with your actual coordinates
P3 = [4, 0]; % Example values, replace with your actual coordinates
% Define the parameter t from 0 to 1
t = linspace(0, 1, 100);
% Compute the Bézier curve for n = 3 using the recursive definition
P0_0 = repmat(P0, length(t), 1); P1_0 = repmat(P1, length(t), 1);
P2_0 = repmat(P2, length(t), 1); P3_0 = repmat(P3, length(t), 1);
P1_1 = (1 - t).' .* P0_0 + t.' .* P1_0; P2_1 = (1 - t).' .* P1_0 + t.' .*
P2_0;
P3_1 = (1 - t).' .* P2_0 + t.' .* P3_0; P2_2 = (1 - t).' .* P1_1 + t.' .*
P2_1;
P3_2 = (1 - t).' .* P2_1 + t.' .* P3_1; P3_3 = (1 - t).' .* P2_2 + t.' .*
P3_2;
% Plot the cubic Bézier curve
figure; plot(P3_3(:,1), P3_3(:,2), 'b', 'LineWidth', 2);
hold on; plot([P0(1) P1(1) P2(1) P3(1)], [P0(2) P1(2) P2(2) P3(2)], 'ro--
'); % Plot control points and polygon
legend('Bézier Curve', 'Control Polygon'); title('Cubic Bézier Curve (n =
3)'); xlabel('x'); ylabel('y'); grid on; axis equal;
```

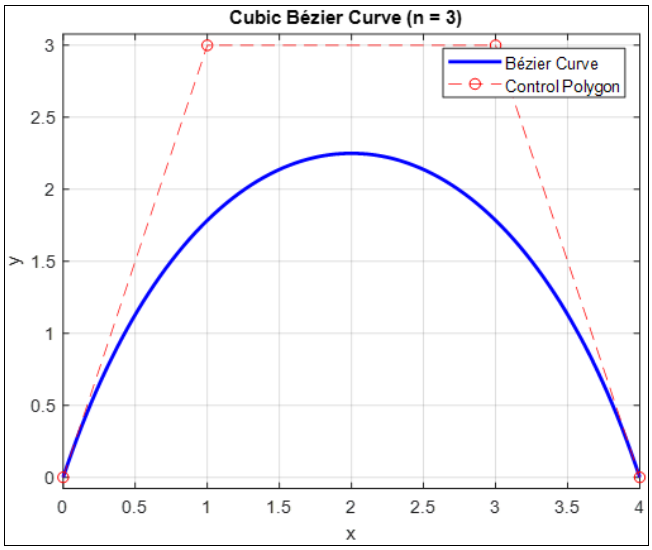

Fig 7: Bézier curve for n = 3 using the recursive definition

2. Properties of the Curve:

The following useful properties are discussed in this study:

1) The weights add up to 1 (they are barycentric). This is easily shown from Newton's binomial theorem $(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i}$:

$$1 = (t + (1-t))^n = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} = \sum_{i=0}^{n} B_{n,i}(t) \quad (10)$$

2) The curve passes through the two endpoints P_0 and P_n. We assume that $0^0 = 1$ and observe that

$$B_{n,0}(0) = \binom{n}{0} 0^0 (1-0)^{n-0} = 1.1.1^n = 1,$$

Which implies

$$P(0) = \sum_{i=0}^{n} P_i B_{n,i}(0) = P_0 B_{n,i}(0) = P_0.$$

Also, the relation

$$B_{n,n}(1) = \binom{n}{n} 1^n (1-1)^{n-n} = 1.1.0^0 = 1,$$

Implies

$$P(1) = \sum_{i=0}^{n} P_i B_{n,i}(1) = P_n B_{n,n}(1) = P_1.$$

3) Another interesting property of the B´ezier curve is its symmetry with respect to the numbering of the control points. If we number the points $P_n, P_{n-1}, \ldots, P_0$, we end up with the same curve, except that it proceeds from right (point P_0) to left (point P_n). The Bernstein polynomials satisfy the identity $B_{n,j}(t) = B_{n,n-j}(1-t)$, which can be proved directly and which can be used to prove the symmetry

$$\sum_{j=0}^{n} P_j B_{n,j}(t) = \sum_{j=0}^{n} P_{n-j} B_{n,j}(1-t)$$

4) The first derivative (the tangent vector) of the curve is straightforward to derive

$$P^t(t) = \sum_{i=0}^{n} P_i \acute{B}_{n,i}(t)$$

$$= \sum_{i=0}^{n} P_i \binom{n}{i} [i t^{i-1} (1-t)^{n-i} + t^i (n-i)(1-t)^{n-i-1}(-1)]$$

$$\sum_{i=0}^{n} P_i \binom{n}{i} i t^{i-1} (1-t)^{n-i} - \sum_{i=0}^{n-1} P_i \binom{n}{i} t^i (n-i)(1-t)^{n-i-1}$$

(Using the identity $n\binom{n-1}{i-1} = i\binom{n}{i}$, we get)

$$n \sum_{i=0}^{n} P_i \binom{n-1}{i-1} t^{i-1} (1-t)^{(n-1)(i-1)} - n \sum_{i=0}^{n-1} P_i \binom{n-1}{i} t^i (n-i)(1-t)^{n-1-i}$$

(But $\binom{n-1}{i-1} t^{i-1} (1-t)^{(n-1)(i-1)} = B_{n-1,i-1}(t)$, so)

$$n \sum_{i=0}^{n-1} P_{i+1} B_{n-1,i}(t) - n \sum_{i=0}^{n-1} P_i B_{n-1,i}(t)$$

$$n \sum_{i=0}^{n-1} [P_{i+1} - P_i] B_{n-1,i}(t)$$

$$n \sum_{i=0}^{n-1} \Delta P_i B_{n-1,i}(t) \quad \text{where} \quad \Delta P_i = P_{i+1} - P_i \quad (11)$$

Note that the tangent vector is a B´ezier weighted sum (of n terms) where each Bernstein polynomial is the weight of a "control point" ΔP_i (ΔP_i is the difference of two points, hence it is a vector, but since it is represented by a pair or a triplet, we can conveniently consider it a point). As a result, the second derivative is obviously another B´ezier sum based on the $n-1$ "control points" $\Delta^2 P_i = \Delta P_{i+1} - \Delta P_i = P_{i+2} - 2P_{i+1} + P_i$.

5) The weight functions $B_{n,i}(t)$ have a maximum at $t = \frac{i}{n}$. To see this, we first differentiate the weights

$$\dot{B}_{n,i}(t) = \binom{n}{i}[it^{i-1}(1-t)^{n-i} + t^i(n-i)(1-t)^{n-i-1}(-1)]$$

$$= \binom{n}{i} it^{i-1}(1-t)^{n-i} - \binom{n}{i} t^i(n-i)(1-t)^{n-1-i}$$

then equate the derivative to zero $\binom{n}{i} it^{i-1}(1-t)^{n-i} - \binom{n}{i} t^i(n-i)(1-t)^{n-1-i} = 0.$ Dividing by $t^{i-1}(1-t)^{n-i-1}$ yields $i(1-t) - t(n-i) = 0$ or $t = \frac{i}{n}$.

6) The two derivatives $P^t(0)$ and $P^t(1)$ are easy to derive from Equation (1*1*) and are used to reshape the curve. They are $P^t(0) = n(P_1 - P_0)$ and $P^t(1) = n(P_n - P_{n-1})$. Since n is always positive, we conclude that $P^t(0)$, the initial tangent of the curve, points in the direction from P_0 to P_1. This initial tangent can easily be controlled by moving point P_1. The situation for the final tangent is similar.

7) The B´ezier curve features global control. This means that moving one control point P_i modifies the entire curve. Most of the change, however, occurs at the vicinity of P_i. This feature stems from the fact that the weight functions $B_{n,i}(t)$ are nonzero for all values of t except $t = 0$ and $t = 1$. Thus, any change in a control point P_i affects the contribution of the term $P_i B_{n,i}(t)$ for all values of t. The behavior of the global control of the B´ezier curve is easy to analyze. When a control point P_k is moved by a vector (α, β) to a new location $P_k + (\alpha, \beta)$, the curve $P(t)$ is changed from the original sum $\sum B_{n,i}(t)P_i$ to

$$\sum_{i=0}^{n} B_{n,i}(t)P_i + B_{n,k}(t)(\alpha,\beta) = P(t) + B_{n,k}(t)(\alpha,\beta)$$

Thus, every point $P(t_0)$ on the curve is moved by the vector $B_{n,k}(t_0)(\alpha,\beta)$. The points are all moved in the same direction, but by different amounts, depending on t_0.(In principle, the figure is for a rational curve, but the particular choice of weights in the figure results in a standard curve.)

8) The concept of the convex hull of a set of points was introduced. Here, we show a connection between the B´ezier curve and the convex hull. Let $P_1, P_2, \ldots, P_n$ be a given set of points and let a point P be constructed as a barycentric sum of these points with nonnegative weights, i.e.,

$$P = \sum_{i=1}^{n} a_i P_i \qquad \text{where} \qquad \sum a_i = 1 \qquad \text{and } a_i \geq 0. \tag{12}$$

It can be shown that the set of all points P satisfying Equation (12) lies in the convex hull of P_1, P_2 through P_n. The B´ezier curve, Equation (5), satisfies Equation (12) for all values of t, so all its points lie in the convex hull of the set of control points. Thus, the curve is said to have the convex hull property. The significance of this property is that it makes the B´ezier curve more predictable. A designer specifying a set of control points needs just a little experience to visualize the shape of the curve, since the convex hull property guarantees that the curve will not "stray" far from the control points.

9) The control polygon of a B´ezier curve intersects the curve at the first and the last points and in general may intersect the curve at a certain number m, of points (Fig (1), where m is 2, 3, or 4, may help to visualize this). If we take a straight segment and maneuver it to intersect the curve as many times as possible, we find that the number of intersection points is always less than or equal m. This property of the B´ezier curve may be termed variation diminution.

10) Imagine that each control point is moved 10 units to the left. Such a transformation will move every point on the curve to the left by the same amount. Similarly, if the control points are rotated, reflected, or are subject to any other affine transformation, the entire curve will be transformed in the same way. We say that the B´ezier curve is invariant under affine transformations. However, the curve is not invariant under projections. If we compute a three-dimensional B´ezier curve and project every point on the curve by a perspective projection, we end up with a two-dimensional curve $P(t)$.

If we then project the three-dimensional control points and compute a two-dimensional B´ezier curve $Q(t)$ from the projected, two-dimensional points, the two curves $P(t)$ and $Q(t)$ will be different. Invariance under projections can be achieved by switching from the standard B´ezier curve to the rational B´ezier curve.

2.1 Reparametrizing the Curve:

The parameter t varies normally in the range $[0,1]$. It is, however, easy to reparametrize the B´ezier curve such that its parameter varies in an arbitrary range $[a, b]$, where a and b are real and $a \leq b$. The new curve is denoted by $P_{ab}(t)$ and is simply the original curve with a different parameter:

$$P_{ab}(t) = P\left(\frac{t-a}{b-a}\right)$$

The two functions $P_{ab}(t)$ and $P(t)$ produce the same curve when t varies from a to b in the former and from 0 to 1 in the latter. Notice that the new curve has tangent vector

$$P^t_{ab}(t) = \frac{1}{b-a}P^t\left(\frac{t-a}{b-a}\right)$$

Reparametrization can also be used to answer the question: Given a B´ezier curve $P(t)$ where $0 \leq t \leq 1$, how can we calculate a curve $Q(t)$ that's defined on an arbitrary part of $P(t)$? More specifically, if $P(t)$ is defined by control points P_i and if we select an interval $[a, b]$, how can we calculate control points Q_i such that the curve $Q(t)$ based on them will go from $P(a)$ to $P(b)$ [i.e., $Q(0) = P(a)$ and $Q(1) = P(b)$] and will be identical in shape to $P(t)$ in that interval? As an example, if $[a, b] = [0, 0.5]$, then $Q(t)$ will be identical to the first half of $P(t)$. The point is that the interval $[a, b]$ does not have to be inside $[0,1]$. We may select, for example, $[a, b] = [0.9, 1.5]$ and end up with a curve $Q(t)$ that will go from $P(0.9)$ to $P(1.5)$ as t varies from 0 to 1. Even though the B´ezier curve was originally designed with $0 \leq t \leq 1$ in mind, it can still be calculated for t values outside this range. If we like its shape in the range $[0.2,1.1]$, we may want to calculate new control points Q_i and obtain a new curve $Q(t)$ that has this shape when its parameter varies in the standard range $[0,1]$. Our approach is to define the new curve $Q(t)$ as $P([b-a]t + a)$ and express the control points Q_i of $Q(t)$ in terms of the control points P_i and a and b. We illustrate this technique with the cubic B´ezier curve. This curve is given by Equation (8) and we can therefore write

$$Q(t) = P([b-a]t + a)$$

$$= (([b-a]t+a)^3, ([b-a]t+a)^2, ([b-a]t+a), 1)\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

$$= (t^3, t^2, t, 1)\begin{pmatrix} (b-a)^3 & 0 & 0 & 0 \\ 3a(b-a)^2 & (b-a)^2 & 0 & 0 \\ 3a^2(b-a) & (b-a) & b-a & 0 \\ a^3 & a^2 & 0 & 1 \end{pmatrix}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

$$= T(t) \cdot A \cdot M \cdot P$$

$$= T(t) \cdot M \cdot M^{-1} \cdot A \cdot M \cdot P$$

$$= T(t) \cdot M \cdot (M^{-1} \cdot A \cdot M) \cdot P$$

$$= T(t) \cdot M \cdot B \cdot P$$

$$= T(t) \cdot M \cdot Q,$$

Where

$$B = M^{-1} \cdot A \cdot M$$

$$= \begin{pmatrix} (1-a)^3 & 3(a-1)^2a & 3(1-a)a^2 & a^3 \\ (a-1)^2(1-b) & (a-1)(-2a-b+3ab) & a(a+2b-3ab) & a^2b \\ (1-a)(-1+b)^2 & (b-1)(-a-2b+3ab) & b(2a+b-3ab) & ab^2 \\ (1-b)^3 & 3(b-1)^2b & 3(1-b)b^2 & b^3 \end{pmatrix} \quad (13)$$

The four new control points $Q_i, i = 0,1,2,3$ are therefore obtained by selecting specific values for a and b, calculating matrix B, and multiplying it by the column

$$P = (P_0, P_1, P_2, P_3)^T.$$

Example (3): We select values $b = 2$ and $a = 1$. The new curve $Q(t)$ will be identical to the part of $P(t)$ from $P(1)$ to $P(2)$ (normally, of course, we don't calculate this part, but this example assumes that we are interested in it). Matrix B becomes, in this case

$$B = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 2 \\ 0 & 1 & -4 & 4 \\ -1 & 6 & -12 & 8 \end{pmatrix}$$

(It is easy to verify that each row sums up to 1) and the new control points are

$$\begin{pmatrix} Q_0 \\ Q_1 \\ Q_2 \\ Q_3 \end{pmatrix} = B \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} P_3 \\ -P_2 + 2P_3 \\ P_1 - 4P_2 + 4P_3 \\ -P_0 + 6P_1 - 12P_2 + 8P_3 \end{pmatrix}$$

To understand the geometrical meaning of these points, we define three auxiliary points R_i as follows:

$$R_1 = P_1 + (P_1 - P_0),$$
$$R_2 = P_2 + (P_2 - P_1),$$
$$R_3 = R_2 + (R_2 - R_1) = P0 - 4P_1 + 4P_2,$$

And write the Q_i's in the form

$$Q_0 = P_3,$$
$$Q_1 = P_3 + (P_3 - P_2),$$
$$Q_2 = Q_1 + (Q_1 - R_2) = P_1 - 4P_2 + 4P_3,$$
$$Q_3 = Q_2 + (Q_2 - R_3) = -P_0 + 6P_1 - 12P_2 + 8P_3.$$

Fig (3) illustrates how the four new points Q_i are obtained from the four original points P_i.

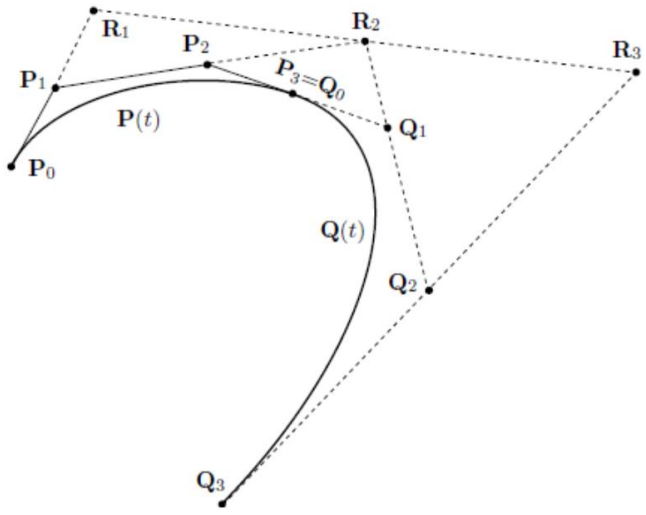

Fig 8: Control Points for the Case [a, b] = [1, 2]

MATLB Code The new curve Q(t) will be identical to the part of P(t) from P(1) to P(2) (normally

```
% Define the original control points (replace these with actual values)
P0 = [0, 0]; % Example values, replace with your actual coordinates
P1 = [1, 2]; % Example values, replace with your actual coordinates
P2 = [3, 3]; % Example values, replace with your actual coordinates
P3 = [4, 0]; % Example values, replace with your actual coordinates
% Define the transformation matrix B
B = [0, 0, 0, 0; 0, 1, -1, 2;  0, 1, -1, 6; -4, 4, -12, 8];
% Calculate the new control points
Q = B * [P0; P1; P2; P3];
% Display the new control points
disp('New control points Q:'); disp(Q);
% Auxiliary points
R1 = P1 + (P1 - P0); R2 = P2 + (P2 - P1); R3 = R2 + (R2 - R1);
% Alternative calculation of Q_i's using auxiliary points
Q0_alt = P3; Q1_alt = P3 + (P3 - P2);
Q2_alt = Q1_alt + (Q1_alt - R2); Q3_alt = Q2_alt + (Q2_alt - R3);
% Display the alternative calculated control points
disp('Alternative calculation of Q:');
disp([Q0_alt; Q1_alt; Q2_alt; Q3_alt]);
% Verify that both methods give the same result
if isequal(round(Q, 10), round([Q0_alt; Q1_alt; Q2_alt; Q3_alt], 10))
    disp('The calculated control points match.');
else
    disp('The calculated control points do not match.');
end
% Define the parameter t from 0 to 1
t = linspace(0, 1, 100);
% Compute the Bézier curve Q(t) using the new control points
Q_curve = (1 - t)'.^3 * Q(1,:) + 3 * (1 - t)'.^2 .* t' * Q(2,:) + 3 * (1
- t)' .* t'.^2 * Q(3,:) + t'.^3 * Q(4,:);
% Plot the original control points and their Bézier curve
figure;
plot(P0(1), P0(2), 'ro', 'MarkerSize', 8, 'DisplayName', 'P_0'); hold on;
plot(P1(1), P1(2), 'go', 'MarkerSize', 8, 'DisplayName', 'P_1');
plot(P2(1), P2(2), 'bo', 'MarkerSize', 8, 'DisplayName', 'P_2');
plot(P3(1), P3(2), 'mo', 'MarkerSize', 8, 'DisplayName', 'P_3');
plot([P0(1), P1(1), P2(1), P3(1)], [P0(2), P1(2), P2(2), P3(2)], 'k--',
'DisplayName', 'Original Control Polygon');
% Plot the new control points and their Bézier curve
plot(Q_curve(:,1), Q_curve(:,2), 'b', 'LineWidth', 2, 'DisplayName',
'Bézier Curve Q(t)');
plot(Q(1,1), Q(1,2), 'r*', 'MarkerSize', 8, 'DisplayName', 'Q_0');
plot(Q(2,1), Q(2,2), 'g*', 'MarkerSize', 8, 'DisplayName', 'Q_1');
plot(Q(3,1), Q(3,2), 'b*', 'MarkerSize', 8, 'DisplayName', 'Q_2');
plot(Q(4,1), Q(4,2), 'm*', 'MarkerSize', 8, 'DisplayName', 'Q_3');
plot([Q(1,1), Q(2,1), Q(3,1), Q(4,1)], [Q(1,2), Q(2,2), Q(3,2), Q(4,2)],
'b--', 'DisplayName', 'New Control Polygon');
% Add legends and titles
legend('show'); title('Original and New Bézier Curve Control Points');
xlabel('x'); ylabel('y'); grid on; axis equal;
```

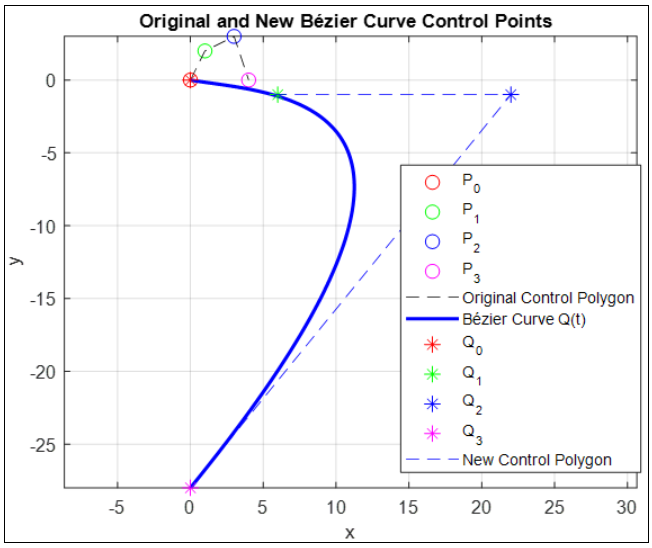

Fig 9: Original and New Bézier Curve Control Points

Example (4): We select $b = 2$ and $a = 0$. The new curve $Q(t)$ will be identical to $P(t)$ from $P(0)$ to $P(2)$. Matrix B becomes

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 1 & -4 & 4 & 0 \\ -1 & 6 & -12 & 8 \end{pmatrix}$$

And the new control points V_i are

$$\begin{pmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{pmatrix} = B \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} P_0 \\ -P_0 + 2P_1 \\ P_0 - 4P_1 + 4P_2 \\ -P_0 + 6P_1 - 12P_2 + 8P_3 \end{pmatrix}$$

And it is easy to see that they satisfy $V_0 = P_0, V_1 = R_1, V_2 = R_3$, and $V_3 = Q_3$.

MTLAB Code of the New Cubic Bézier Curve Q(t)

```
% Define the original control points (replace these with actual values)
P0 = [0, 0]; % Example values, replace with your actual coordinates
P1 = [1, 2]; % Example values, replace with your actual coordinates
P2 = [3, 3]; % Example values, replace with your actual coordinates
P3 = [4, 0]; % Example values, replace with your actual coordinates
% Define the transformation matrix B
B = [1, 0, 0, 0; -1, 2, 0, 0; 1, -4, 0, 0; 4, -12, 0, 8];
% Calculate the new control points
V = B * [P0; P1; P2; P3];
% Display the new control points
disp('New control points V:'); disp(V);
% Auxiliary points
R1 = P1 + (P1 - P0); R2 = P2 + (P2 - P1);
R3 = R2 + (R2 - R1); Q3 = -P0 + 6*P1 - 12*P2 + 8*P3;
% Alternative calculation of V_i's using auxiliary points
V0_alt = P0; V1_alt = R1;V2_alt = R3;V3_alt = Q3;
% Display the alternative calculated control points
disp('Alternative calculation of V:');
disp([V0_alt; V1_alt; V2_alt; V3_alt]);
% Verify that both methods give the same result
if isequal(round(V, 10), round([V0_alt; V1_alt; V2_alt; V3_alt], 10))
    disp('The calculated control points match.');
else
    disp('The calculated control points do not match.');
end
% Define the parameter t from 0 to 1
t = linspace(0, 1, 100);
% Compute the Bézier curve Q(t) using the new control points
Q_curve = (1 - t)'.^3 * V(1,:) + 3 * (1 - t)'.^2 .* t' * V(2,:) + 3 * (1
- t)' .* t'.^2 * V(3,:) + t'.^3 * V(4,:);
% Plot the new cubic Bézier curve
figure; plot(Q_curve(:,1), Q_curve(:,2), 'b', 'LineWidth', 2);
hold on; plot([V(1,1) V(2,1) V(3,1) V(4,1)], [V(1,2) V(2,2) V(3,2)
V(4,2)], 'ro--'); % Plot control points and polygon
legend('Bézier Curve', 'Control Polygon');
title('New Cubic Bézier Curve Q(t)');
xlabel('x'); ylabel('y'); grid on; axis equal;
```

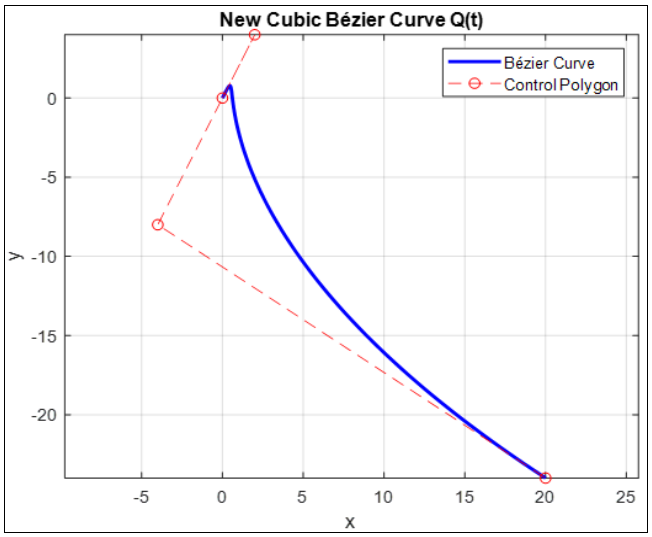

Fig 10: Original and New Bézier Curve Control Points

Results

The most important results i can summarize as follows:

1. t is used in the linear curve association to express the distance ($B(t)$ on the line between the points $P_0$and P_1. For example, when $t = 0.25$, ($B(t)$ is the first quarter of the distance between P_0 and P_1. t changes between the values 0 and 1, while $B(t)$ describes the straight line between P_0 and P_1.
2. In a second-order Bezier curve, we can create intermediate points such as Q_0 and Q_1. These points have a value located in the interval $[1,0]$: The first point ($Q_0(t)$ lies between P_0 and P_1 and describes the linear Bezier curve between these two points. The second point ($Q_1(t)$ lies between P_1 and P_2 and describes the linear Bezier curve between these two points The point ($B(t)$ lies on the line connecting the two points ($Q_0(t)$ and $(Q_1(t)$) and describes a second-order Bezier curve.
3. To obtain a Bezier curve for higher degrees, note that the idea in arriving at drawing the curve is to divide each line by placing a point on it and then connect the new points to each other (note that we get rid of a point at each stage), and we repeat the process until we have only one line left. So, we put ($B(t)$ on it and then we draw the curve.

Recommendations

The most important recommendations i can summarize as follows:

1. It is noted that the curve lies completely within the convex closure of the points, so these points are used with ease to form the required curve. Therefore, it is desirable to use Bezier curves widely in computer graphics in order to draw smooth and smooth curves.
2. It is also possible to apply geometric transformations (such as rotation and sliding) to the curve by applying this transformation to the control points of this curve.
3. In animation applications, Bezier curves are used in drawing such as drawing motion pictures. The user draws the path using a Bezier curve, and the application undertakes the task of preparing for the movement of this element along the previously drawn path. Likewise, in 3D graphics, the Bezier curve draws 3D paths.

References

1. Senay Baydas, Bluent Karakas. Defining a curve as a Bezier curve. Journal of Taibah University for Science, 2019.
2. Salma Naseer, others. A Class of Sextic Trigonometric Bezier Curve with Two Shape Parameters, researchgate, 2021.
3. Mridula Dube, Bharti Yadav. The Quintic Trigonometric Bézier Curve with single Shape Parameter. International Journal of Scientific and Research Publications, 2014.
4. Ferdous Sohel, others. A dynamic Bezier curve model. Semantic scholar, 2005.
5. Aslı Ayar, Bayram Sahin. Curves used in highway design and Bezier curves, doi.org, 2022.
6. Jiwoong Choi, *et al*. Smooth Path Generation Based on B´ezier Curves for Autonomous Vehicles, iaeng.org, 2009.
7. David S. Curves and Surfaces for Computer Graphics. New York: Springer, 2006.