The Representation, Generation and Industry Application of Bézier Curves

Tynan Purdy

002129-0002

IB Extended Essay

Mathematics

February 2019

Words: 2592

Abstract

Curves are everywhere; in architecture, in objects, and in books. At some point, many of those curves existed on a computer. Graphic designs, 3D models of products, paper documents, so much of our carpentered world is rendered on a computer before it ever sees the third dimension. Computer-Aided Design (CAD) programs have taken over the manufacturing industry with versatile and extensive software packages that simulate any physical object right on the computer. Graphic design programs are at the source of every print and digital display. But these computer programs must be able to produce curved lines and surfaces in a computationally efficient way, or else producing all the curves used every day would bring computer hardware to its knees. A French automotive engineer by the name of Pierre Bézier implemented an efficient and easily adjustable method of digital curve rendering called the Bézier curve. His method of curve drawing has made its way into countless digital mediums, including 3D and 2D design software, digital typefaces, and more.

Words: 169

Contents

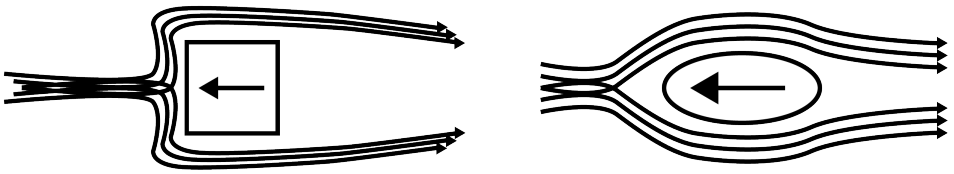
1 Introduction

Bézier curves, a mathematical process for drawing curves, has found its way into the lifeblood of modern communication, digital graphics, and text. Every screen in the world is almost constantly using Bézier curves to draw whatever it is displaying. Most technology users do not know what Bézier curves are, despite looking at them every day for hours on end. The lines of text on this very page were drawn using Bézier curves. The art on billboards was rendered with Bézier curves before they were ever printed. The ability to render virtual text characters instantaneously has contributed to the content boom of the internet. The following pages will describe what Bézier curves are, where they came from, and exactly how they are used to deliver words and shapes to the world.

The mathematics behind Bézier curves existed for decades before computers were invented. It was the French automotive engineer, Pierre Bézier (1910-1999), who popularized the method for digital curve and surface drawing. Bézier was a mathematician and head of design at Renault, a Paris based automaker (**?**). Before the origin of these special curves is discussed, it must be understood why curves are important and so difficult to procure digitally. Curves have been used in mechanical design for hundreds of years, and for good reason. They give a smooth, attractive form that is not only more inviting to the touch than a sharp edge, but also lends itself toward

aerodynamics. Curves are perfect for effortlessly passing through fluids such as air and water, as demonstrated in Figure **??**.

Figure 1: Fluid is redirected harshly around a cornered object relative to a rounded object.



(a) Cornered object (b) Rounded object

Unsurprisingly, aerodynamics in air and water accounts for the primary uses of curved surfaces both in ancient Rome and today. Consider the cars seen on the road today. Even an astute observer would have a hard time finding a perfectly flat surface or hard corner anywhere on a vehicle. Geometric patterns are so popular in the 21^{st} century, yet they have not made their way to cars. Curves are so energy efficient compared to straight lines and corners that there is no compelling reason to abandon them.

Curves are also much safer than square edges and corners. Even objects that appear to have square corners often have a small fillet to smooth them out and avoid potential accidental cuts or scratches. The safe properties of curves arise from the same traits as its aerodynamic properties. An angled edge is more able to split open a solid that a curved edge (hence why knives must be sharp to function properly). The concern of safety against hard

edges is addressed extensively in the manufacturing space. A wide variety of tools exists purely to reduce sharp edges to a smooth and safe curve.

Curved edges and surfaces clearly have their benefits. As discussed, there are ways to add curves to a cornered object, but it would be better to include curves in the original design. With manually operated tools, creating a curve is not particularly difficult. The human brain can visualize a shape and instruct the hands to use tools to make any shape or surface imaginable that the tool is capable of creating. Handcrafted wood objects can be curved with a chisel and some sanding. In the car manufacturing industry, most auto bodies are modeled at full scale in clay during the design phase. Handcrafting is sufficient for many purposes, but when the end goal of a design is to mass-produce it, at some point the shape of the object must be described by a computer. Manually operated tools are time consuming and labor intensive to make thousands of identical parts for mass production, which is not feasible for large scale manufacturing. Computer numerical control (CNC) tools have solved the mass production issue by allowing tools to be programmed to execute precise operations precisely and repeatably. CNC tools require far less attention from a human operator and produce consistent machining, perfect for mass production. The catch is, since the machine is computer controlled, the computer must be able to describe the cuts it is supposed to make, which means they must be described mathematically. Pierre Bézier was one of the first engineers to address the need for digital representation of

physical parts, and so began development of the Bézier curve for Renault, his French automotive employer. Bézier was not the first, nor the only engineer to work on a mathematical curve algorithm for computers, but his was the widely adopted method. The famous De Casteljau's algorithm was developed for Citroën, a competing auto company in Paris. His algorithm was kept secret for years after its development in the '60s (**?**), but later became the dominant method of producing Bézier curves on a computer.

2 Bézier Curves

A Bézier curve is comprised of two major components: control points and Bernstein polynomials. The final curve is produced by a summation of Bernstein polynomials with the control point coordinate as a coefficient of each polynomial term. The Bernstein portion of the formula will be discussed before the complete De Casteljau algorithm.

2.1 Bernstein Polynomials

Bernstein polynomials are the expression that represent Bézier curves digitally. Before understanding Bézier curves mathematically, Bernstein polynomials must be defined. Bernstein functions are a method of approximating more complex polynomials. Linear polynomial terms are defined by Equation **??**. Coefficients for each of the linear terms are given by the binomial coefficient formula (Equation **??**).

$$B_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i} \qquad i = 0, ..., n \tag{1}$$

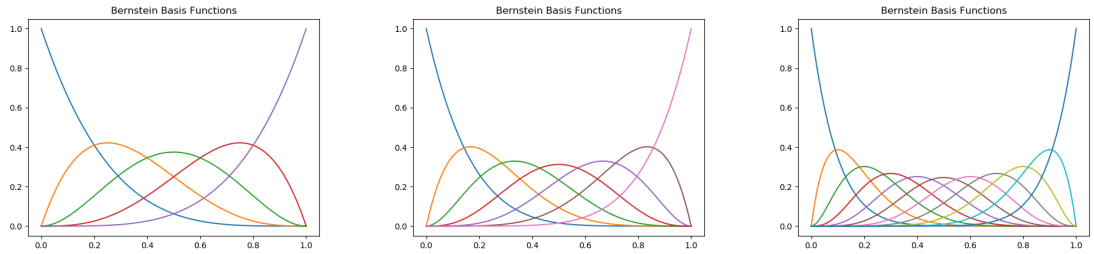
n: degree of the Bernstein function

i: iteration, or term

$$\binom{n}{i} = \prod_{i=1}^{n} \frac{n+1-i}{i} \tag{2}$$

Like any function, a Bernstein function of the degree n will have $n+1$ many terms. The graphs below show the value of each individual term from $t = 0$ to $t = 1$ given different orders of the Bernstein function.

Figure 2: Bernstein basis function graphs (Listing **??**)



(a) $n = 4$ (b) $n = 6$ (c) $n = 10$

The summation of the terms in a Bernstein polynomial is always equal to 1. Bernstein polynomials are used to approximate more complicated curved functions. Because the Bernstein function can create any shaped curve, it is well suited for this use case.

2.2 Parametric Bézier Curves

A Bézier curve is a linear combination of Bernstein polynomials, each with coefficients corresponding to the coordinates of control points P_i. These control points are not necessarily intersected by the curve but they do determine its shape.

$$F_n(t) = \sum_{i=0}^{n} P_i B_{n,i}(t) \tag{3}$$

Because the algorithm in Equation **??** allows for any degree n of Bézier curve, with any number of control points, Listing **??** was able be used to generate all of the Bézier curve figures in this section. With fewer than 50 lines of python code, any Bézier curve can be calculated and graphed.

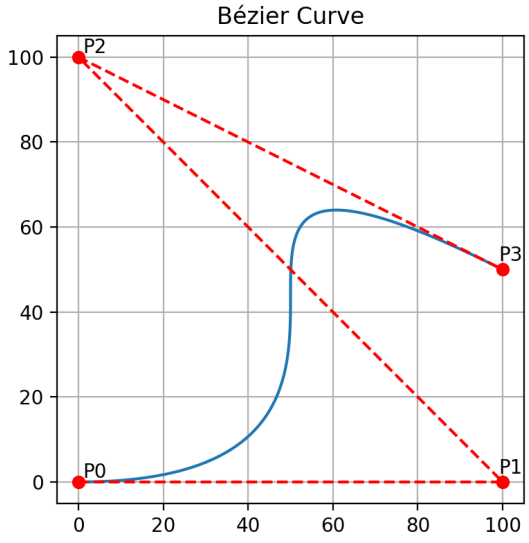


Figure 3: Bézier curve with 4 control points.

Consider Figure **??**, a curve with four control points: P_0, P_1, P_2, and P_3. P_0 and P_3 are the endpoints of the curve, so they will be intersected by the curve. P_1 and P_2 will not be intersected by the curve. They do 'pull' the curve in their direction. Placing P_1 closer to a line between P_0 and P_3 will reduce the bend towards P_1.

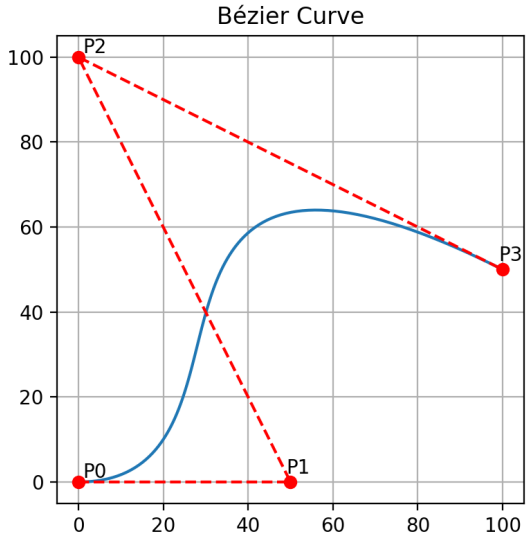


Figure 4: Demonstration of how control points influence the curve. The closer the control points are to forming a line, the less curvature.

The behavior of the curve in relation to its control points comes from the properties of the equation that defines the curve. Bézier curves are defined by a parametric combination of Bernstein polynomials. While traditional graphed functions use an x-coordinate to solve for a y-coordinate, a parametric defines x and y individually as functions of time, t. Take Equation **??** below. The parametric equation uses the parameter t and two control points, P_0 and P_1. The equation is solved given t from 0 to 1. At $t = 0$, the result is

equal to P_0, and at $t = 1$ it is equal to P_1.

$$F(t) = P_0 + t(P_1 - P_0) \tag{4}$$

In order to graph Equation **??** as a linear Bézier curve (ignore the 'linear curve' paradox, it works the same way), two control points must be designated, for instance $P_0(1, 1)$ and $P_1(2, 3)$. Then, the coordinates will be substituted into the parametric equation for x and y.

$$x(t) = 1 + t(2 - 1)$$

$$y(t) = 1 + t(3 - 1)$$

Evaluating $x(t)$ and $y(t)$ from 0 to 1 produces coordinate points on the 'curve' shown in Figure **??**.

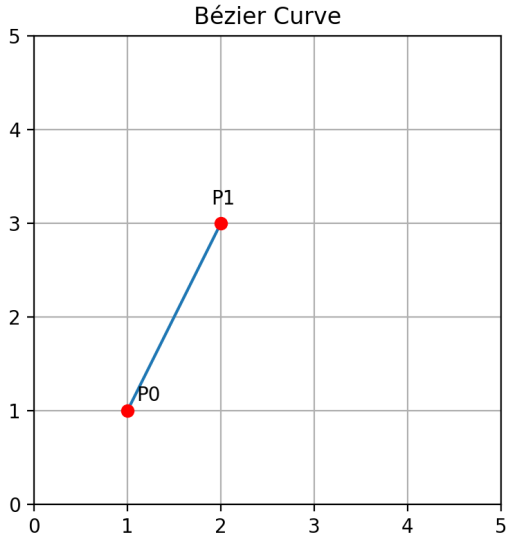


Figure 5: Linear Bézier curve.

As t increases from 0 to 1, the coordinates produced become further from P_0 and closer to P_1 because P_0 is getting subtracted away and P_1 is being added at equal rates.

Drawing a simple line with a parametric equation is not particularly impressive. However, when the equation is rearranged to be a Bernstein polynomial, the formula can easily be adapted to accommodate more dimensions. Equation **??** is a 1^{st} order Bernstein polynomial with the control points as coefficients if rearranged like below.

$$F(t) = (1-t)P_0 + tP_1 \tag{5}$$

Each term in the Bernstein polynomial is matched sequentially with a

control point as a coefficient to create formulas for higher order polynomials (See Table **??**).

$B_{n,i}(t)$	Bernstein Term	CP Coefficient	Complete Term
$B_{1,0}$	$1-t$	P_0	$(1-t)P_0$
$B_{1,1}$	t	P_1	tP_1
$B_{2,0}$	$(1-t)^2$	P_0	$(1-t)^2P_0$
$B_{2,1}$	$2(1-t)t$	P_1	$2(1-t)tP_1$
$B_{2,2}$	t^2	P_2	t^2P_2
$B_{3,0}$	$(1-t)^3$	P_0	$(1-t)^3P_0$
$B_{3,1}$	$3(1-t)^2t$	P_1	$3(1-t)^2P_1$
$B_{3,2}$	$3(1-t)t^2$	P_2	$3(1-t)t^2P_2$
$B_{3,3}$	t^3	P_3	t^3P_3

Table 1: Bernstein polynomial, order 1 to 3, terms and corresponding Bézier control point coefficients.

For instance, below is a 2^{nd} order Bernstein polynomial, which uses 3 control points.

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

With 3 control points, the resulting graph forms a curve.

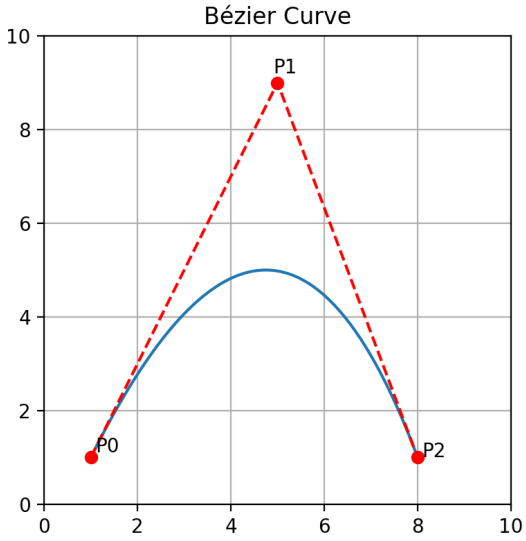


Figure 6: Bézier curve with 3 control points, resulting from a 2nd order Bernstein polynomial.

One byproduct of how the curve is formed is that the curve always lies completely within the area bounded by the control points. The curve in Figure **??** lies completely within the triangle formed by P_0, P_1, and P_2.

2nd order Bernstein polynomial Bézier curves are referred to as quadratic Bézier curves. Another common form is the cubic Bézier curve, which uses a 3rd order Bernstein polynomial.

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

As can be expected, a 3rd order polynomial has 4 terms and 4 control points.

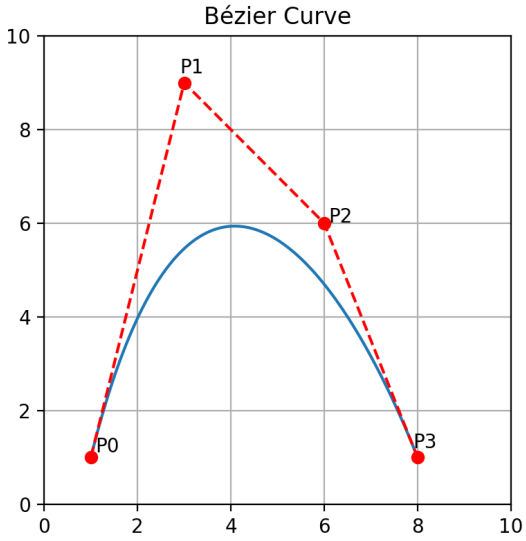


Figure 7: Cubic Bézier curve with 4 control points, generated with a 3rd order Bernstein polynomial.

All of the Bézier graphs shown in this section (Figures **??**, **??**, **??**, **??**, and **??**) were generated by the python script in Listing **??**.

A specific coordinate at any time t can be solved for in one function with a summation of the Bernstein terms multiplied by their Bézier control point coefficient, as shown in Equation **??**. This formula is known as De Casteljau's algorithm, and is the formula used by computers as they render Bézier curves (Listing **??**).

3 Applications

3.1 Fonts

Until laser printers had a high enough dot density to produce legible text, digital fonts were virtually unnecessary and nonexistent. As printer technology developed, a digital typeface became a more and more likely development. Bézier curves ultimately answered the question of how to produce a digital text character with its easy to use and efficient to encode curves.

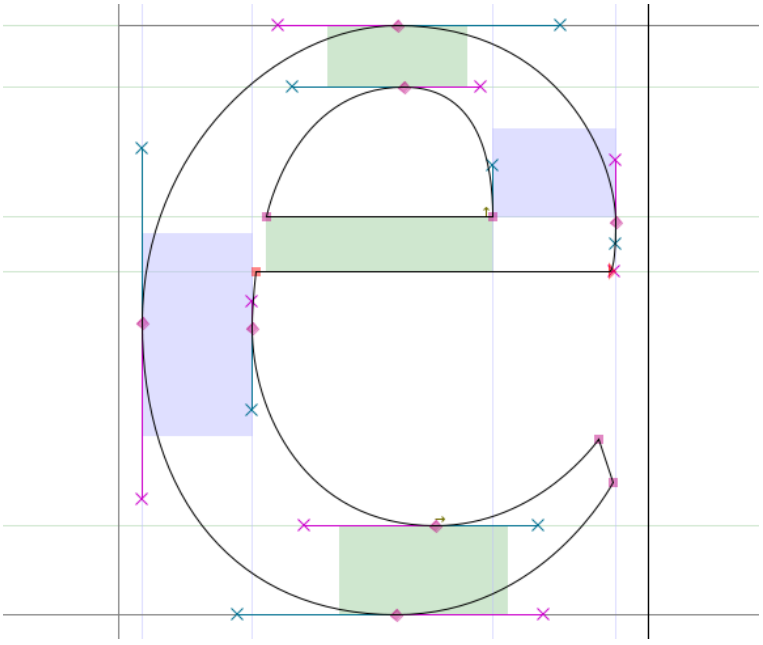


Figure 8: Letter 'e' with visible Bézier control points (**?**, 9)

Bézier curves are suited well for fonts not only because computers can handle them well, but their adaptability and ease of adjustment lends toward the process of optical balancing. The design process of a font, even individual characters, requires very close attention to optical balance to make characters

look and feel the way they are meant to be. Optimizing fonts is a topic for another discussion, so for now Bézier curves offer the flexibility needed to create quality fonts while also being efficient for computers to render.

Computational efficiency is incredibly important for font rendering simply because text is rendered so frequently. As can be seen in the python code in Listing **??**, rendering a curve is a very simple task that only requires a few numerical arguments. As the primary method of displaying information in language form on a screen, text is rendered all the time. The efficiency of the Bézier curve allows text to not be a heavy burden on the CPU to render.

The prevalence of Bézier curves in fonts has to do with the main digital font formats which are industry standards: TrueType and PostScript. PostScript was designed by Adobe in the early '80s as their core technology. It is a programming language dedicated toward document rendering. Fonts, as well as vector graphics, were drawn as a series of lines and cubic Bézier curves. These documents could be scaled to any resolution, and therefore were compatible with virtually any laser printer.

Apple developed the TrueType outline font format to replace bitmap fonts in its Macintosh computers. TrueType was released with Macintosh System 7. It uses lines and quadratic Bézier curves. While quadratic curves are not as flexible as cubic curves with one less control point, they are computationally lighter for the same reason.

TrueType managed to become the dominant font standard after Apple licensed it to Microsoft. TrueType was added to the Windows operating system in 1991, and Microsoft published many TrueType fonts. Later on Microsoft created the OpenType format, which borrowed much of the underlying structure from TrueType, but included features for more control over typefaces.

Both TrueType and PostScript replaced bitmap fonts entirely. Bitmap fonts were much harder to scale and less elegant than the curve-based successors. A bitmap font had to be recreated individually for each resolution, which was useful for making legibility adjustments at different sizes but was much more labor intensive to scale up or down for different sizes. Vector fonts could be automatically scaled, which is part of why they became the industry standard in digital typefaces. Now, on both Windows and macOS, TrueType, OpenType, and PostScript fonts are supported.

3.2 Vectors

PostScript's use of cubic Bézier curves also made its way to the dominant vector graphic design software, Adobe Illustrator®. Illustrator relies heavily on cubic Bézier curves to draw most every shape. The system is based entirely around the control points of the curves. The endpoints of curves are referred to as control points, while the two points that lie off the curve are called handles. The control point-based control scheme has been adopted

as the default method of interaction universally in vector drawing software, and can be found in Illustrator and other popular vector programs such as CorelDRAW and Inkskape.

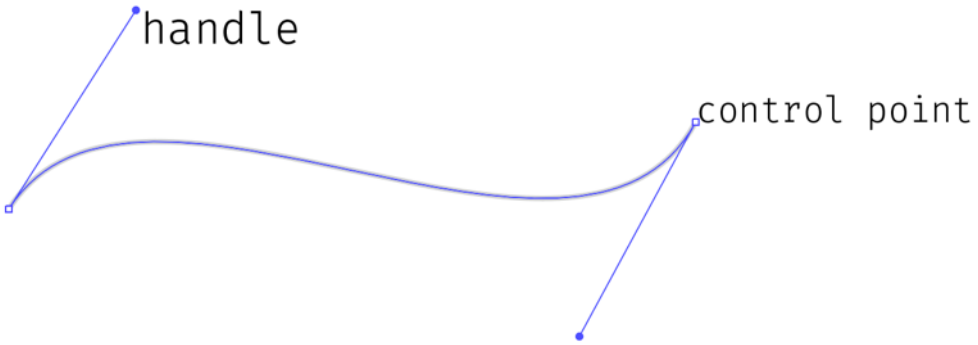


Figure 9: Example curve with control points visible, drawn the pen tool in Adobe Illustrator®.

Vector design programs are a necessary tool for the creation of any digital graphic (websites, magazines, logos, brochures, advertisements, etc.).

The Scalable Vector Graphics (SVG) file format is the common vector file format which can be read by mainstream vector software packages. It is written in eXtensible Markup Language (XML), a self-descriptive markup language meant to be legible for computers as well as humans. The SVG is an open standard format by the World Wide Web Consortium, used for graphics. SVGs are compatible with vector based graphic design programs, and can be integrated in-line with HTML. They are perfect for web use because the files are small, simple text files, but have the benefits of being infinitely scalable

via the properties of vectors, hence the name Scalable Vector Graphic. For web use, this is ideal, since there are a wide range of screen resolutions to render web content on, and keeping large raster images on a web server can quickly become slow and cumbersome for a client that is loading the site. Of course, for a standard, vector based file, Bézier curves are the perfect way to take care of curves, shapes and fonts rendered in SVG files.

4 Summary

The intuitive nature of Bézier curves makes them perfect for artists and designers, but also the computational efficiency of Bernstein polynomials reduces load on the GPU of any machine that is rendering vector graphics. The perfect match of user and system benefits has resulted in widespread industry adoption in vector rendering and vector file formats. In today's climate of popular geometric and vector graphic design, as well as the ongoing and expanding use of print and digital text, Bézier curve technology is integral infrastructure for artists, designers, and readers around the world.

5 Appendix

List of Figures

List of Tables

Listing 1: Bézier Curve

```
from numpy import linspace as lin  # linespace
from scipy.special import comb  # binomial coefficient
import matplotlib.pyplot as plt  # plotting

cpx = [0.0, 0.5, 2.0, 1.5] # control point x coordinates
cpy = [0.0, 3.0, 0.0, 3.0] # control point y coordinates

t = lin(0, 1, 100)  # 100 sample inputs between 0 and 1
num = 3  # degree 0 indexed

# returns the value from the Bernstein basis function at the point x
def bern(i, n, t):
    return comb(n, i) * t ** i * (1 - t) ** (n-i)

# Generalized De Casteljau's Explicit formula
def f(a, t):
    ret = 0
    for i in range(num + 1):
        ret += a[i] * bern(i, num, t)
    return ret

def x(t): return f(cpx, t) # implemented on X's control points
def y(t): return f(cpy, t) # implemented on Y's control points

plt.plot([x(i) for i in t], [y(i) for i in t]) # plot bezier curve
plt.plot(cpx, cpy, 'ro') # plot control points
plt.plot(cpx, cpy, 'r:') # graph lines between control points
for i in range(num+1):
    # label control points
    plt.text(cpx[i]+0.05, cpy[i]+0.05, 'P{}'.format(i))
plt.title('Bezier') # title
```

```
plt.show() # display graphs
```

Listing 2: Bernstein Basis Functions

```
from numpy import linspace as lin  # linespace
from scipy.special import comb  # binomial coefficient
import matplotlib.pyplot as plt  # plotting

x = lin(0, 1, 100)  # 100 sample inputs between 0 and 1
num = 4  # degree 0 indexed

# returns the value from the Bernstein basis function at the point x
def bern(i, n, t):
    return comb(n, i) * t ** i * (1 - t) ** (n-i)

# iterates through all of the functions for the degree
for c in range(num + 1):
    # plots all of the functions
    plt.plot(x, [bern(c, num, i) for i in x])

plt.title("Bernstein Basis Functions")  # Title
plt.show()  # displays graphs
```