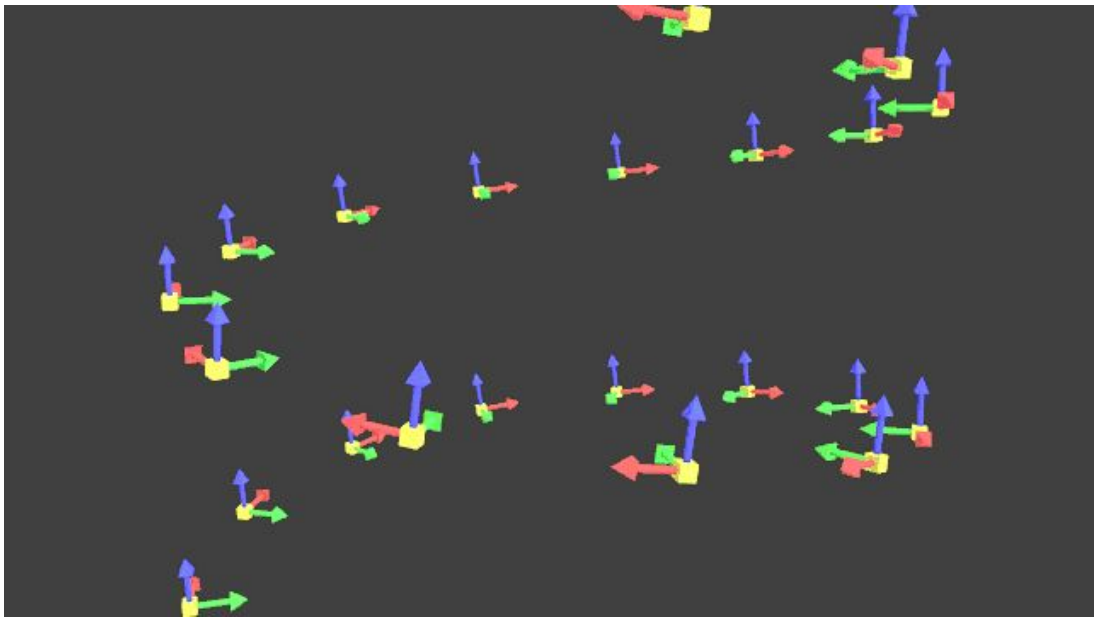
Advances in Robotics and Control

Assignment 3

Design Of Curves

Nivedita Rufus, 2019702002



Question 1

Introduction

The Frenet-Serret frame or sometimes referred to as simply Frenet frame or TNB frame is constructed purely from the velocity and the acceleration of the curve. The velocity is simply described by the first derivative of x(t) i.e. x'(t) and acceleration by the second derivative x''(t). The single directions of the frame are then described by the tangential vector T, normal vector N and binormal vector B with the following equations,

$$\vec{T} = \frac{\vec{x}'(t)}{\left\|\vec{x}'(t)\right\|}$$

$$\vec{B} = \frac{\vec{x}'(t) \times \vec{x}''(t)}{\left\|\vec{x}'(t) \times \vec{x}''(t)\right\|}$$

$$\vec{N} = \vec{B} \times \vec{T}$$

Part A

Consider the curve given by f(t)=(t, t^2, t^3). Write a code to plot the curve and draw a Frenet frame (T, N, B) at any given parameter value.

Steps of Approach

- Define a parameter time and compute the value of the curve given by ***f(t)*** in the question.
- Calculate the value of the first, second and third derivative which will be required to calculate the T, N, B values at the given parameter. These steps are followed in the given code blocks:

```
def make_param(time):
      x_time = time
      y_time = time**2
      z_time = time**3

      dx, dy, dz = first_derivative(time)
      d_dx, d_dy, d_dz = second_derivative(time)
      d_ddx, d_ddy, d_ddz = third_derivative(time)

      r = np.array([x_time, y_time, z_time])
      r_dot = np.array([dx, dy, dz])
      r_ddot = np.array([d_dx, d_dy, d_dz])
      r_dddot = np.array([d_ddx, d_ddy, d_ddz])

      return x_time, y_time, z_time, r, r_dot, r_ddot, r_dddot
```

Calculation of derivatives:

```
def first_derivative(time):
      dx = 1
      dy = 2*time
      dz = 3*(time**2)
      return dx, dy, dz


def second_derivative(time):
      d_dx = 0
      d_dy = 2
      d_dz = 6*time
      return d_dx, d_dy, d_dz


def third_derivative(time):
      d_ddx = 0
      d_ddy = 0
      d_ddz = 6
      return d_ddx, d_ddy, d_ddz
```

- Now the T, N, B values are calculated in accordance with the formulae given below,

$$\vec{T} = \frac{\vec{x}'(t)}{\left\|\vec{x}'(t)\right\|}$$

$$\vec{B} = \frac{\vec{x}'(t) \times \vec{x}''(t)}{\left\|\vec{x}'(t) \times \vec{x}''(t)\right\|}$$

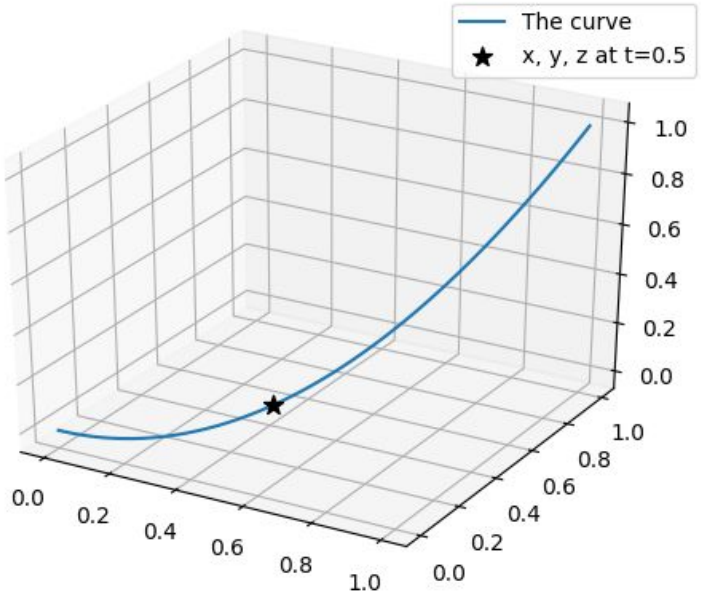
$$\vec{N} = \vec{B} \times \vec{T}$$

These formulae are used to calculate T, N, B values at the given parameter in the following code block:

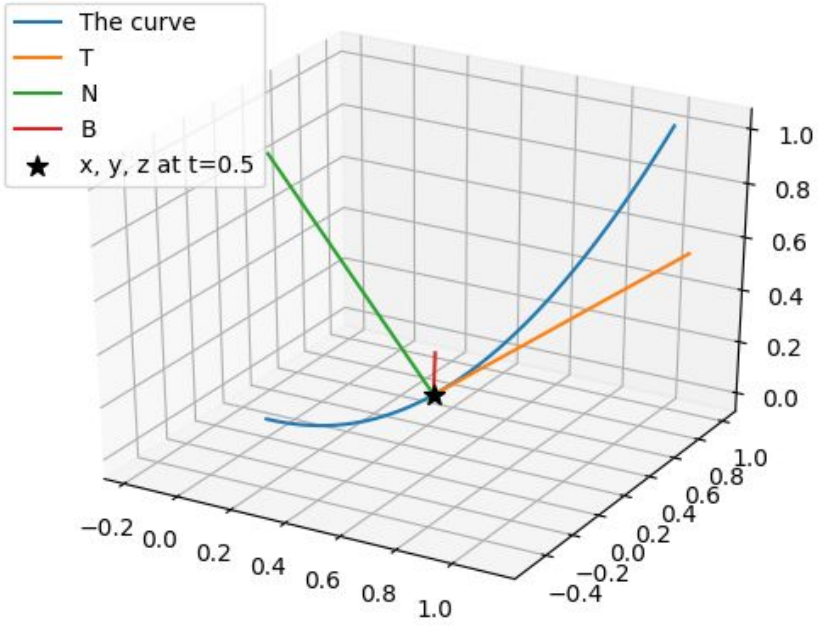
```
def TNB(r, r_dot, r_ddot, r_dddot):
      r_dot_norm = np.linalg.norm(r_dot)
      T = r_dot/r_dot_norm

      B = np.cross(r_dot, r_ddot)
      B_norm = np.linalg.norm(B)
      B = B/B_norm

      N = np.cross(B, T)

      return T, N, B
```

Results

The curve and the value at t=0.5



The T, N, B values at t=0.5



Part B

Plot the Osculating circle at the parameter.

Steps of Approach

- To plot the osculating circle, the center and the curvature of the circle are required.
- The curvature is given by the equation below and is also done in the code block following the equation:

$$\kappa = \left|\frac{dT}{ds}\right| = \frac{|v \times a|}{|v|^3}$$

```
def get_curvature(r, r_dot, r_ddot, r_dddot):
      curvature = np.linalg.norm(np.cross(r_dot, r_ddot))
      c_norm = (np.linalg.norm(r_dot)**3)
      curvature = curvature/c_norm
      return curvature
```

- The center of the circle is calculated in the following code block

```
def get_center(r, N, curvature):
      center = r - (N/curvature)
      return center
```

- And the parametric equation of the osculating circle at time t is given by :

circle_x = centre_x + cos(t)(1/κ)T_x + sin(t)(1/κ)N_x

circle_y = centre_y + cos(t)(1/κ)T_y+ sin(t)(1/κ)N_y

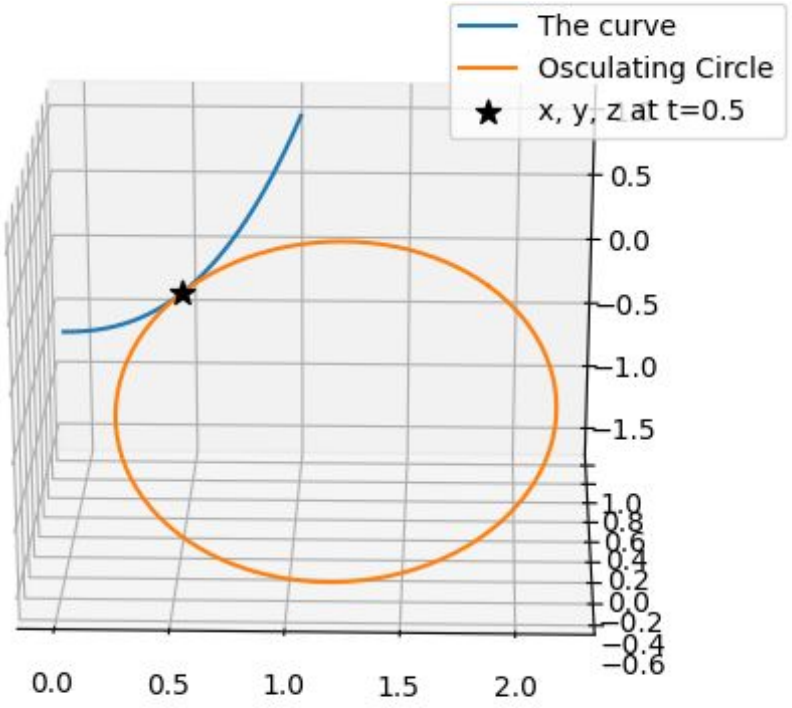
circle_z = centre_z + cos(t)(1/κ)T_z + sin(t)(1/κ)N_z

Where, ***<name>_x, <name>_y, <name>_z*** denote the ***x, y, z*** coordinate of ***<name>*** respectively, κ is the curvature, and ***T, N*** are the tagential and normal values.

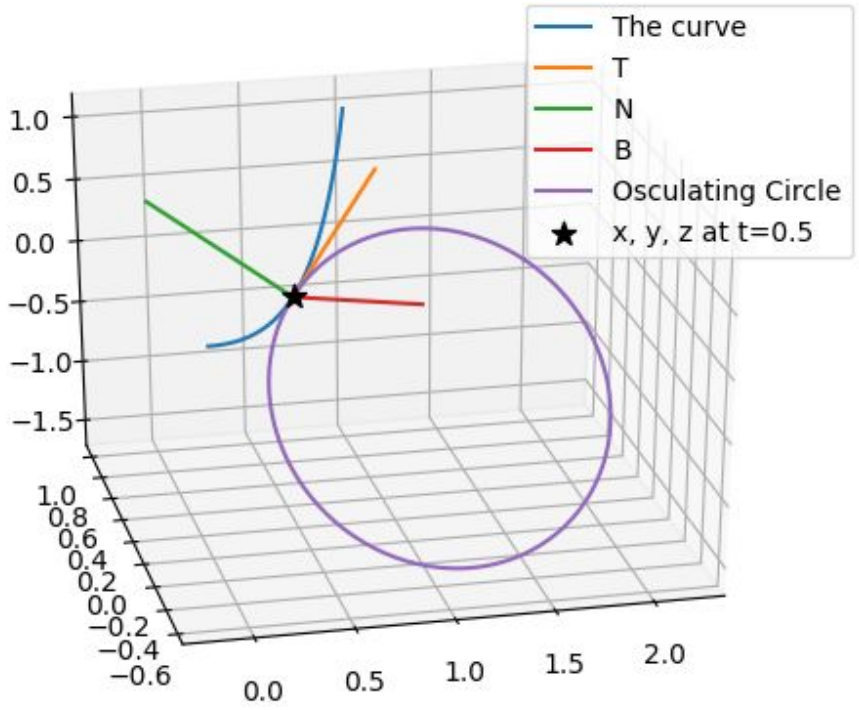
```
x_c = center[0] + np.cos(rad)*(1/curvature)*T[0] +
np.sin(rad)*(1/curvature)*N[0]
      y_c = center[1] + np.cos(rad)*(1/curvature)*T[1] +
np.sin(rad)*(1/curvature)*N[1]
      z_c = center[2] + np.cos(rad)*(1/curvature)*T[2] +
np.sin(rad)*(1/curvature)*N[2]
```

Results

The curve and the osculating circle at t=0.5



The curve, (T, N, B) values and the osculating circle at t=0.5



Part C

Find the Curvature and Torsion of the curve at that parameter value.

Steps of Approach

- The curvature is calculated as mentioned in the previous section,

```
def get_curvature(r, r_dot, r_ddot, r_dddot):
      curvature = np.linalg.norm(np.cross(r_dot, r_ddot))
      c_norm = (np.linalg.norm(r_dot)**3)
```

```
    curvature = curvature/c_norm
    return curvature
```

- The value of torsion is calculated by the following formula,

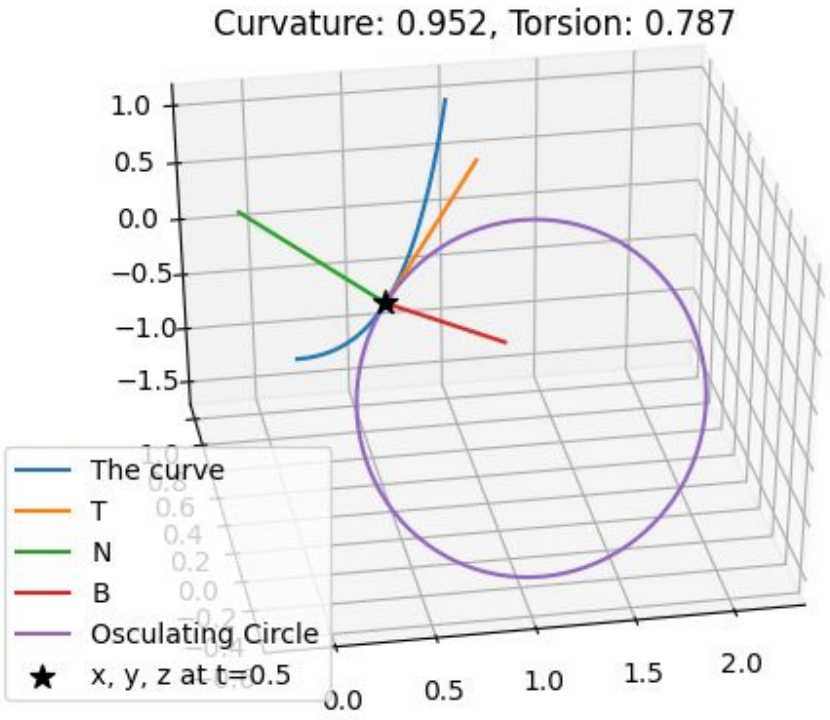
$$|\tau| = \left|\frac{dB}{ds}\right|$$

The same is done in the following code block:

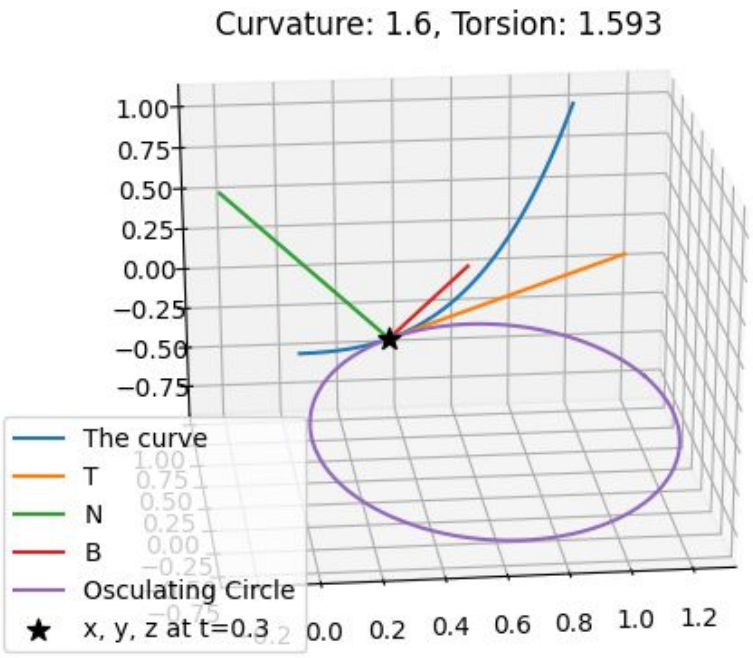
```
def get_torsion(r, r_dot, r_ddot, r_dddot):
    t_norm = (np.linalg.norm(np.cross(r_dot, r_ddot))**2)
    torsion = np.dot(np.cross(r_dot, r_ddot),r_dddot)
    torsion = torsion/t_norm
    return torsion
```

Results

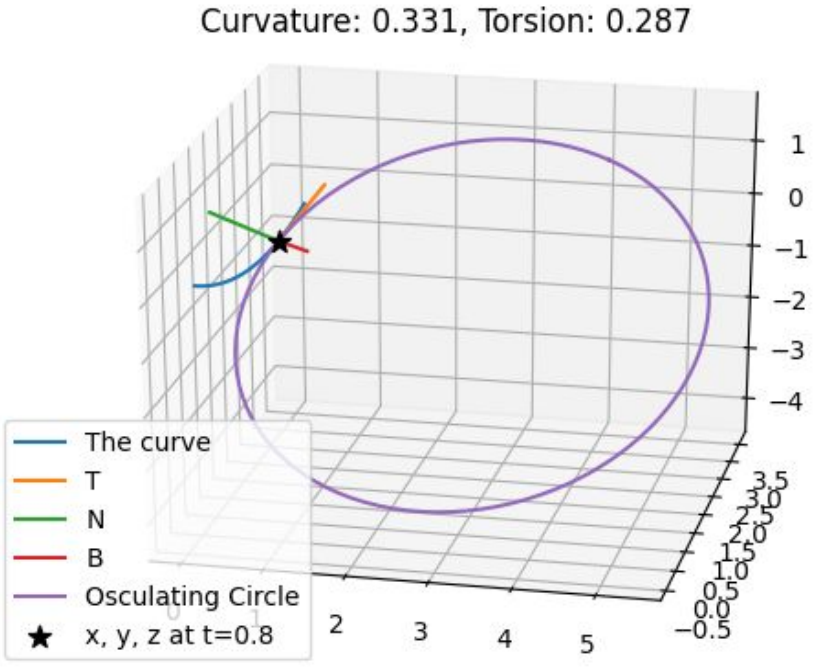
The curvature and torsion values at t=0.5



The curvature and torsion values at t=0.3



The curvature and torsion values at t=0.8



Question 2

Introduction

Bezier curve is discovered by the French engineer Pierre Bézier. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as,

$$\sum_{k=0}^{n} P_i B_i^n(t)$$

Where, P_i is the set of points and ***n*** is the polynomial degree, ***i*** is the index, and ***t*** is the variable.

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

Properties of Bezier Curves

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less than the number of defining polygon points. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the endpoints is the same as that of the vector determined by the first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.

- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

Part A

Write a code to draw a 2D Bezier curve (along with control polygon), where the points will be given randomly within some bounded workspace.

Steps of Approach

- Define random control points in a 2D bounded workspace.
- Calculate the Bezier path using the Bernstein coefficients and the given control points. This is given in the following code block:

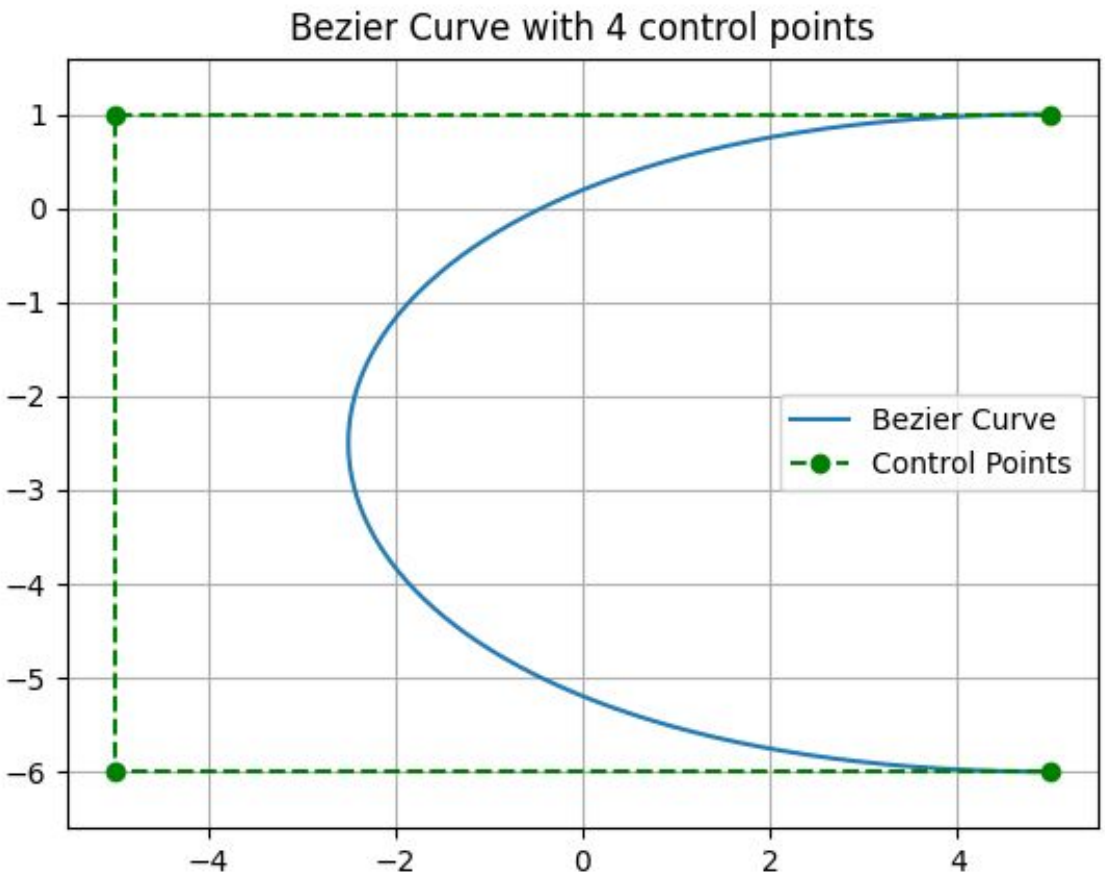
```
def bezier_path(control_points, n_points=100):
    traj = []
    n = len(control_points) - 1
    time_steps = np.linspace(0, 1, n_points)

    for t in time_steps:
        temp = np.sum([bernstein(n, i, t) * control_points[i] for i in
range(0, n + 1)], axis=0)
        traj.append(temp)
    traj = np.array(traj)

    return traj
```

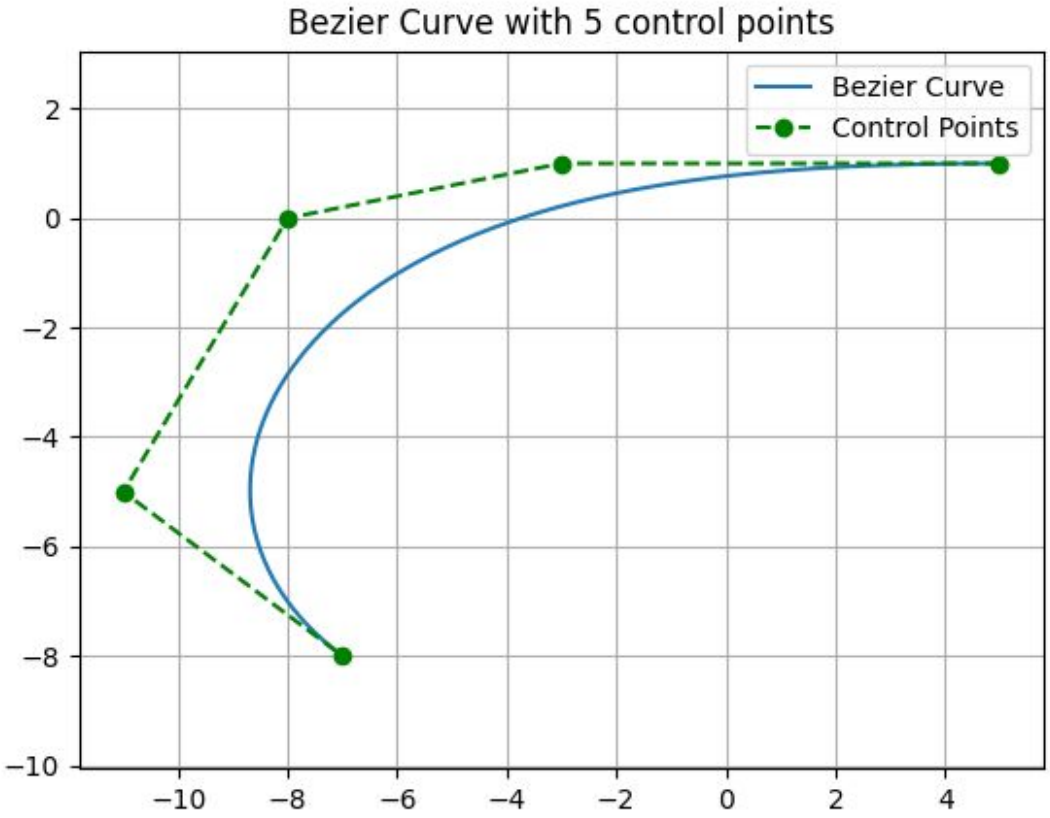
The Bernstein values are calculated using the function given in the following code block:

```
def bernstein(n, i, t):
    return scipy.special.comb(n, i) * t ** i * (1 - t) ** (n - i)
```

- The generated path is now plotted. The experiments were done using different number of control points.

Results





Part B

Select a parameter value u = c to subdivide the Bezier curve, and draw control polygons for individual components.

The meaning of subdividing a curve is to cut a given Bézier curve at ***u = c*** for some u into two curve segments, each of which is still a Bézier curve. Because the resulting Bézier curves must have their own new control points, the original set of control points is discarded. Moreover, since the original Bézier curve of degree ***n*** is cut into two pieces, each of which is a subset of the original degree ***n*** Bézier curve, the resulting Bézier curves must be of degree ***n***.

Steps of Approach

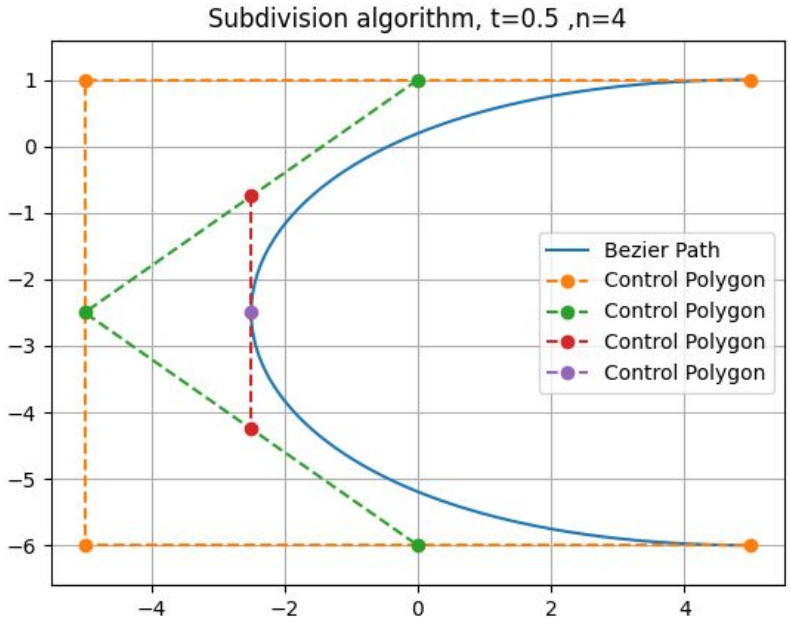
- This problem to subdivide the Bezier Curve is solved using De Casteljau's algorithm.
- The new control points can be calculated using the equation,

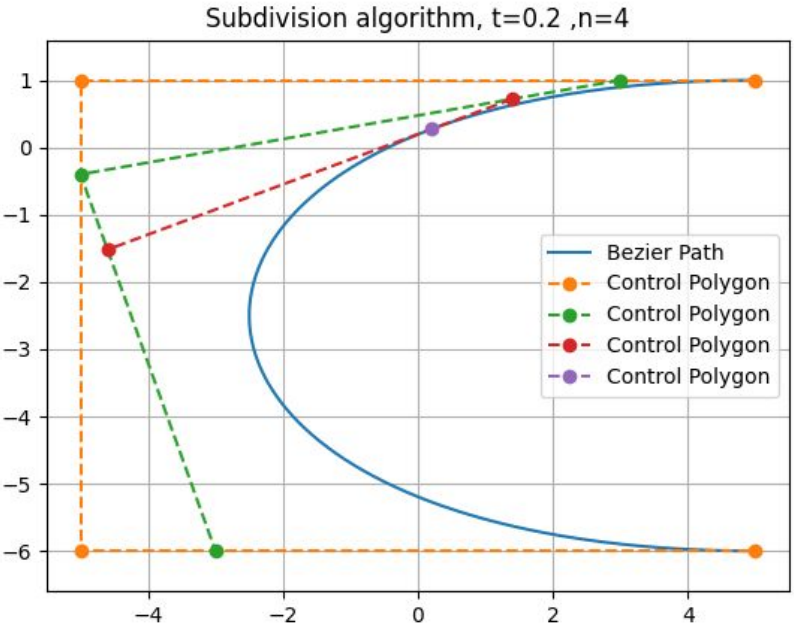
$$\mathbf{b}_i^k(t) = (1-t)\mathbf{b}_i^{k-1} + t\mathbf{b}_{i+1}^{k-1}$$

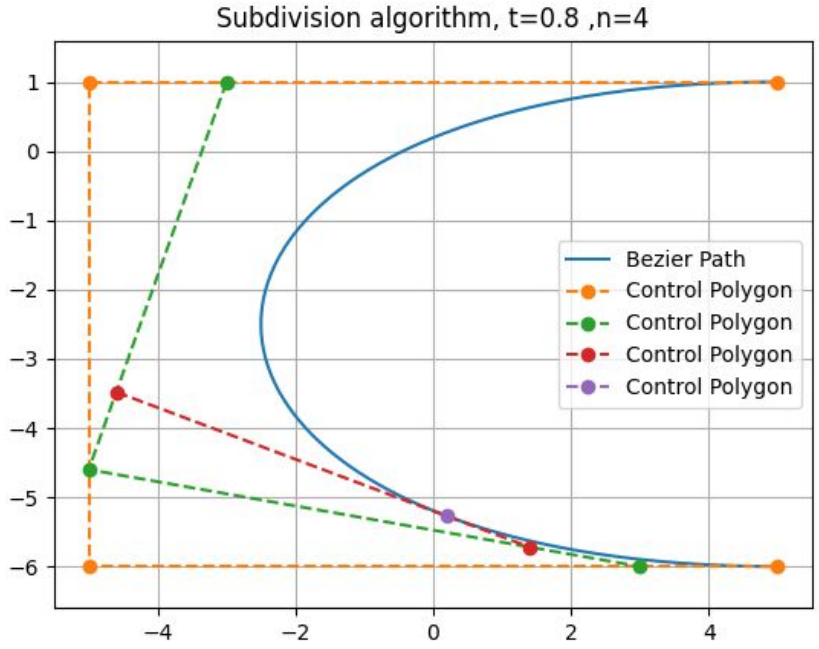
Where, ***k = 1, 2,...n*** and ***i = 0, 1, ..., n - k***. This is also done in the code block given below:

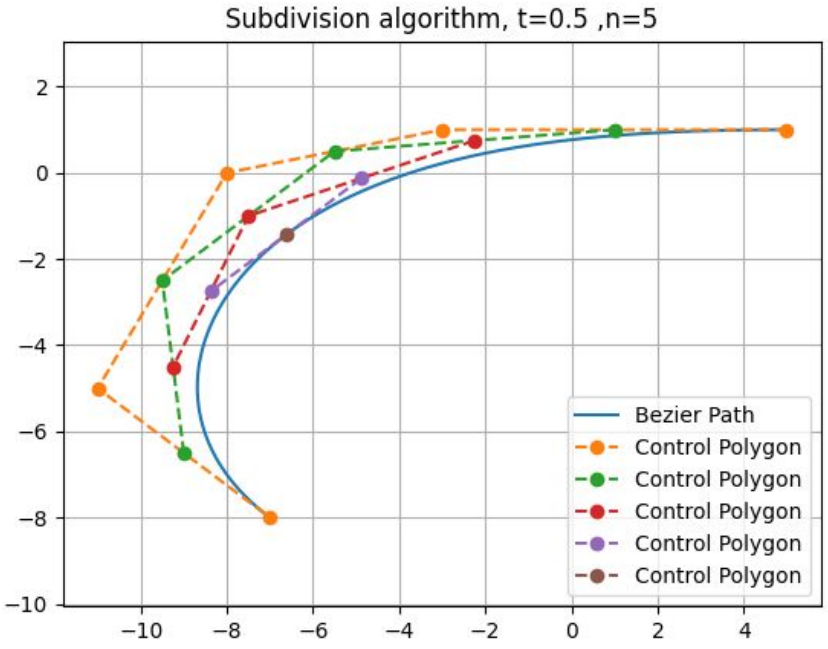
```
def subdivide_curve(points, t=0.5):
    n = len(points)
    for k in range(1,n):
        newpoints = []
        for i in range(0, n-k):
            x = (1-t) * points[i][0] + t * points[i+1][0]
            y = (1-t) * points[i][1] + t * points[i+1][1]
            newpoints.append((x,y))
        newpoints = np.array(newpoints)
        ax.plot(newpoints.T[0], newpoints.T[1], '--o', label = "Control
Polygon")
        points = newpoints
```

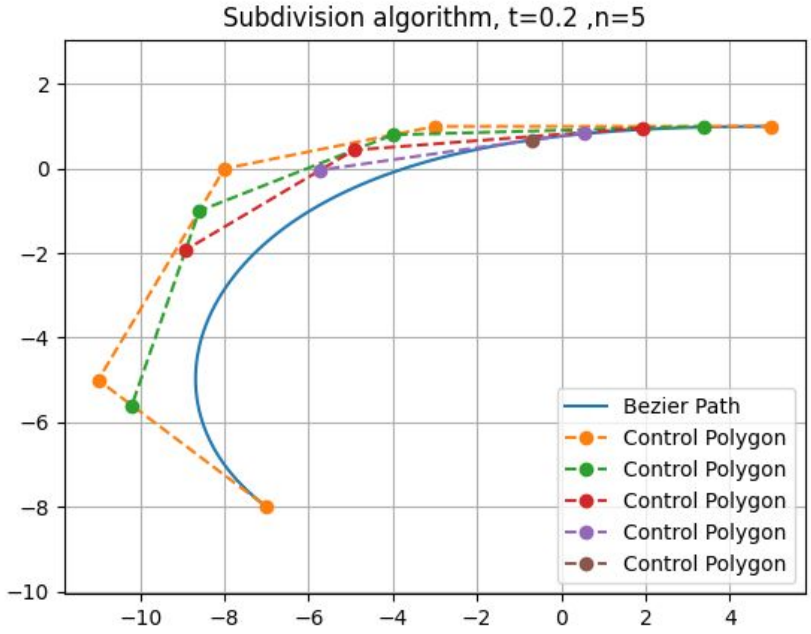
Results

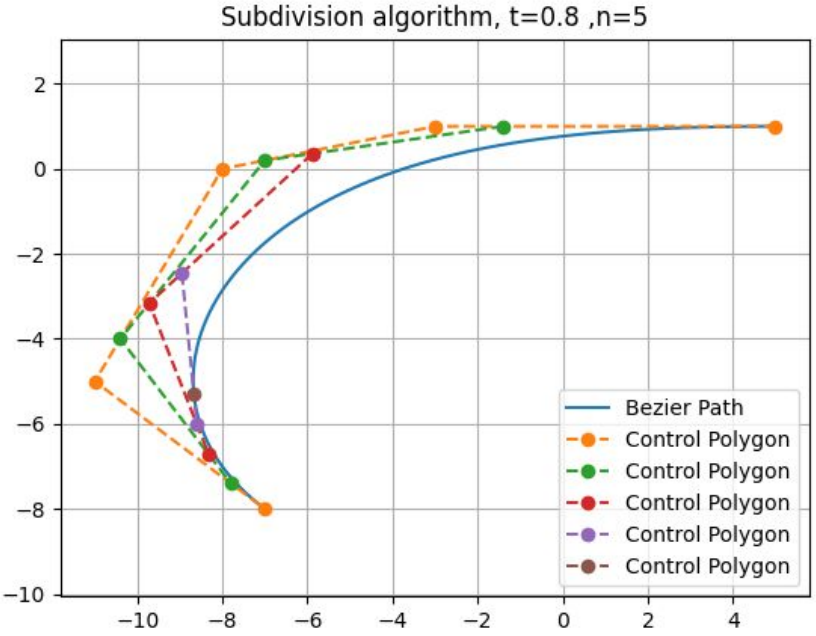

Subdivision algorithm, t=0.5 ,n=4
Bezier Path
Control Polygon
Control Polygon
Control Polygon
Control Polygon
1
0
−1
−2
−3
−4
−5
−6
−4
−2
0
2
4


Subdivision algorithm, t=0.2 ,n=4
Bezier Path
Control Polygon
Control Polygon
Control Polygon
Control Polygon
1
0
−1
−2
−3
−4
−5
−6
−4
−2
0
2
4


Subdivision algorithm, t=0.8 ,n=4
Bezier Path
Control Polygon
Control Polygon
Control Polygon
Control Polygon


Subdivision algorithm, t=0.5 ,n=5
Bezier Path
Control Polygon
Control Polygon
Control Polygon
Control Polygon
Control Polygon


Subdivision algorithm, t=0.2 ,n=5
Bezier Path
Control Polygon
Control Polygon
Control Polygon
Control Polygon
Control Polygon


Subdivision algorithm, t=0.8 ,n=5
Bezier Path
Control Polygon
Control Polygon
Control Polygon
Control Polygon
Control Polygon