```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
```

normally called the "knot sequence". The $N_{i,k}$ functions are described as follows

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } u \in [t_i, t_{i+1}), \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

and if $k > 1$,

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \tag{2}$$

- and $t \in [t_{k-1}, t_{n+1})$.

```
# B-spline basis function (recursive Cox-de Boor)
def de_boor_cox_basis(i, k, t, knots):
    """
    Recursively computes the B-spline basis function B(i, k) using the De Boor-Cox formula.
    Parameters:
    i : int Index of the basis function.
    k : int Degree of the B-spline.
    x : float The parameter value at which to evaluate.
    knot : np.ndarray The knot vector.
    Returns: float The computed basis function value B(i, k).
    """
    if k == 1:
        return 1.0 if knots[i] <= t < knots[i + 1] else 0.0
    denom1 = knots[i + k - 1] - knots[i]
    denom2 = knots[i + k] - knots[i + 1]

    term1 = 0.0
    term2 = 0.0
    if denom1 > 0:
        term1 = ((t - knots[i]) / denom1) * \
            de_boor_cox_basis(i, k - 1, t, knots)
    if denom2 > 0:
        term2 = ((knots[i + k] - t) / denom2) * \
            de_boor_cox_basis(i + 1, k - 1, t, knots)

    return term1 + term2
```

The B-Spline Curve – Analytical Definition

A B-spline curve $\mathbf{P}(t)$, is defined by

$$\mathbf{P}(t) = \sum_{i=0}^{n} \mathbf{P}_i N_{i,k}(t)$$

where

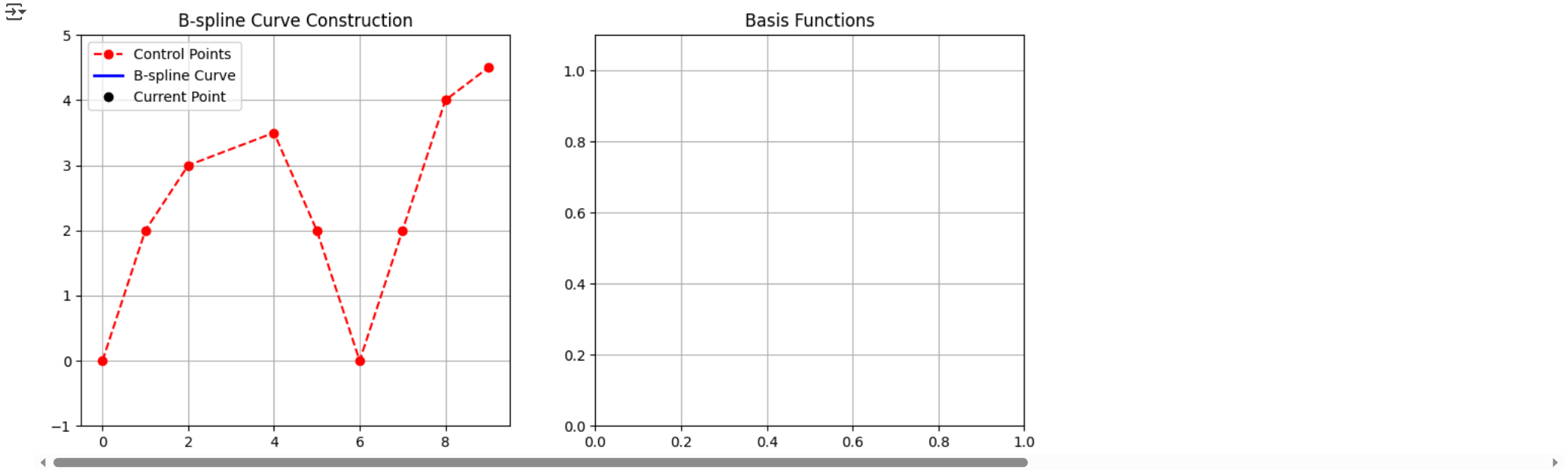
- the $\{\mathbf{P}_i : i = 0, 1, ..., n\}$ are the control points,
- k is the order of the polynomial segments of the B-spline curve. Order k means that the curve is made up of piecewise polynomial segments of degree $k - 1$,
- the $N_{i,k}(t)$ are the "normalized B-spline blending functions". They are described by the order k and by a non-decreasing sequence of real numbers

$$\{t_i : i = 0, ..., n + k\}.$$

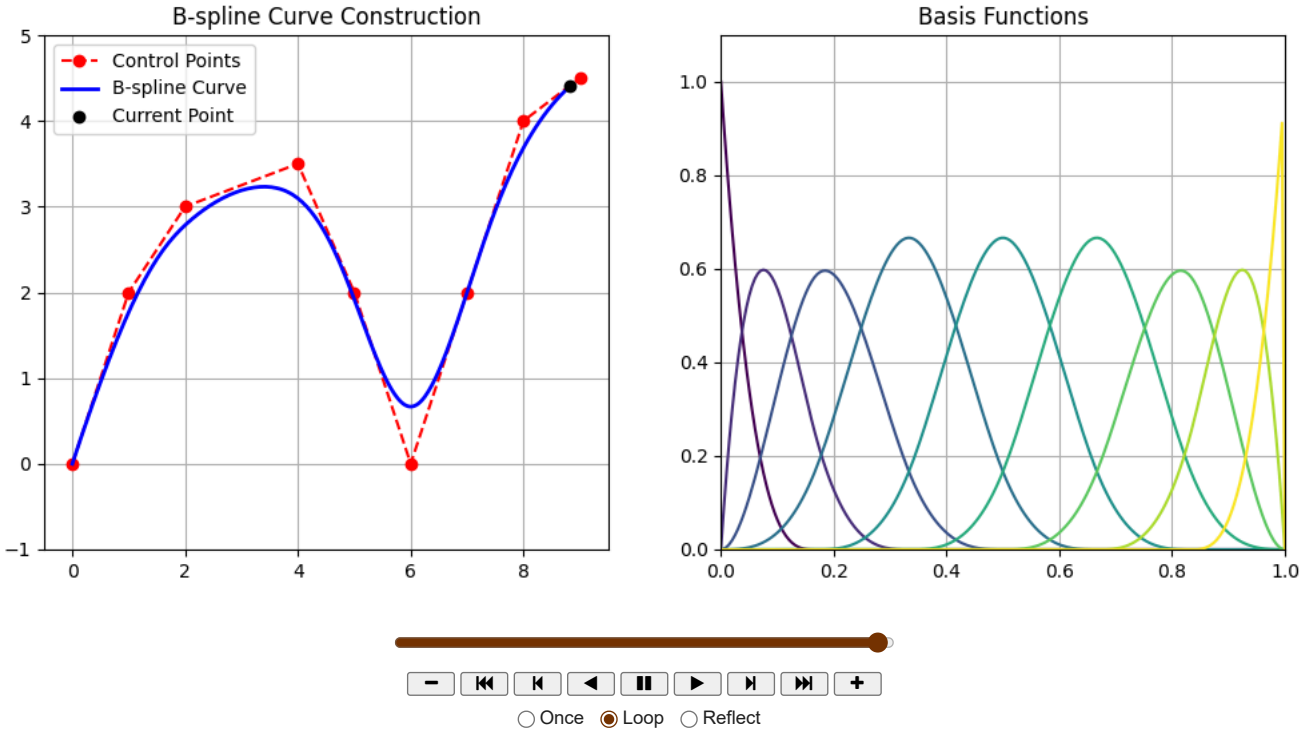
```
# B-spline curve computation
def bspline_point(t, control_points, k, knots):
    n = len(control_points)
    point = np.zeros(2)
    for i in range(n):
        b = de_boor_cox_basis(i, k, t, knots)
        point += b * control_points[i]
    return point
```

```
# Sample control points
control_points = np.array([
    [0, 0],
    [1, 2],
    [2, 3],
    [4, 3.5],
    [5, 2],
    [6, 0],
    [7, 2],
    [8, 4],
    [9, 4.5],
])
n = len(control_points)
```

```
degree = 3  # cubic B-spline

# Clamped knot vector
knots = np.concatenate((
    np.zeros(degree),
    np.linspace(0, 1, n - degree + 1),
    np.ones(degree)
))

# t values for animation
t_vals = np.linspace(knots[degree], knots[-degree-1], 200)


# Setup plot
fig, ax = plt.subplots(1, 2, figsize=(12, 5))
curve_ax = ax[0]
basis_ax = ax[1]

curve_ax.set_title("B-spline Curve Construction")
basis_ax.set_title("Basis Functions")

curve_ax.plot(control_points[:, 0], control_points[:,
              1], 'ro--', label="Control Points")
curve_line, = curve_ax.plot([], [], 'b-', lw=2, label="B-spline Curve")
moving_point, = curve_ax.plot([], [], 'ko', label="Current Point")

colors = plt.cm.viridis(np.linspace(0, 1, n))
basis_lines = [basis_ax.plot([], [], color=colors[i], label=f"N{i},{degree}")[
    0] for i in range(n)]

curve_ax.legend()
basis_ax.set_xlim(knots[degree], knots[-degree-1])
basis_ax.set_ylim(0, 1.1)
curve_ax.set_xlim(-0.5, 9.5)
curve_ax.set_ylim(-1, 5)
basis_ax.grid(True)
curve_ax.grid(True)
```




```
# Increase the animation embed limit
plt.rcParams['animation.embed_limit'] = 50  # You can adjust this value as needed (in MB)


computed_points = []

def animate(frame):
    t = t_vals[frame]
    # degree + 1 because k is order (degree + 1)
    point = bspline_point(t, control_points, degree+1, knots)
    computed_points.append(point)

    if len(computed_points) > 1:
        curve_line.set_data(*np.array(computed_points).T)

    moving_point.set_data([point[0]], [point[1]])

    # Update basis functions
    for i, line in enumerate(basis_lines):
        ts = np.linspace(knots[degree], knots[-degree-1], 200)
        bs = [de_boor_cox_basis(i, degree + 1, ti, knots)
              for ti in ts]  # degree + 1 here as well
        line.set_data(ts, bs)

    return [curve_line, moving_point] + basis_lines


ani = FuncAnimation(fig, animate, frames=len(t_vals), interval=50, blit=True)
plt.tight_layout()

# Display animation in Colab
HTML(ani.to_jshtml())
```




```
<Figure size 640x480 with 0 Axes>
```

```
# Compute and print curve points
print(f"Computed Curve Points: {len(t_vals)} in total")
for t in t_vals:
    point = bspline_point(t, control_points, degree+1, knots)
    print(f"t = {t:.3f} --> point = ({point[0]:.4f}, {point[1]:.4f})")
```

```
Computed Curve Points: 200 in total
t = 0.000 --> point = (0.0000, 0.0000)
t = 0.005 --> point = (0.0891, 0.1769)
t = 0.010 --> point = (0.1756, 0.3457)
t = 0.015 --> point = (0.2595, 0.5069)
t = 0.020 --> point = (0.3410, 0.6605)
t = 0.025 --> point = (0.4202, 0.8068)
t = 0.030 --> point = (0.4971, 0.9461)
t = 0.035 --> point = (0.5718, 1.0784)
t = 0.040 --> point = (0.6445, 1.2041)
t = 0.045 --> point = (0.7153, 1.3234)
t = 0.050 --> point = (0.7842, 1.4365)
t = 0.055 --> point = (0.8513, 1.5436)
t = 0.060 --> point = (0.9167, 1.6449)
t = 0.065 --> point = (0.9806, 1.7407)
t = 0.070 --> point = (1.0429, 1.8311)
t = 0.075 --> point = (1.1039, 1.9165)
```

```
t = 0.080 --> point = (1.1636, 1.9969)
t = 0.085 --> point = (1.2222, 2.0727)
t = 0.090 --> point = (1.2796, 2.1440)
t = 0.095 --> point = (1.3360, 2.2111)
t = 0.101 --> point = (1.3915, 2.2741)
t = 0.106 --> point = (1.4462, 2.3334)
t = 0.111 --> point = (1.5002, 2.3891)
t = 0.116 --> point = (1.5536, 2.4414)
t = 0.121 --> point = (1.6064, 2.4906)
t = 0.126 --> point = (1.6589, 2.5369)
t = 0.131 --> point = (1.7110, 2.5805)
t = 0.136 --> point = (1.7628, 2.6215)
t = 0.141 --> point = (1.8146, 2.6604)
t = 0.146 --> point = (1.8663, 2.6972)
t = 0.151 --> point = (1.9180, 2.7321)
t = 0.156 --> point = (1.9699, 2.7655)
t = 0.161 --> point = (2.0221, 2.7975)
t = 0.166 --> point = (2.0745, 2.8283)
t = 0.171 --> point = (2.1275, 2.8581)
t = 0.176 --> point = (2.1808, 2.8871)
t = 0.181 --> point = (2.2345, 2.9150)
t = 0.186 --> point = (2.2885, 2.9420)
t = 0.191 --> point = (2.3429, 2.9679)
t = 0.196 --> point = (2.3975, 2.9928)
t = 0.201 --> point = (2.4523, 3.0167)
t = 0.206 --> point = (2.5073, 3.0394)
t = 0.211 --> point = (2.5624, 3.0611)
t = 0.216 --> point = (2.6176, 3.0815)
t = 0.221 --> point = (2.6729, 3.1008)
t = 0.226 --> point = (2.7282, 3.1189)
t = 0.231 --> point = (2.7834, 3.1358)
t = 0.236 --> point = (2.8386, 3.1513)
t = 0.241 --> point = (2.8936, 3.1657)
t = 0.246 --> point = (2.9485, 3.1786)
t = 0.251 --> point = (3.0032, 3.1903)
t = 0.256 --> point = (3.0577, 3.2006)
t = 0.261 --> point = (3.1119, 3.2094)
t = 0.266 --> point = (3.1658, 3.2169)
t = 0.271 --> point = (3.2193, 3.2229)
t = 0.276 --> point = (3.2724, 3.2274)
```