YOU_a4q1

November 14, 2021

1 A4-Q1: MySpline

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

1.1 MySpline

[2]:
```
def MySpline(x, y):
    '''
     S = MySpline(x, y)

     Input:
       x and y are arrays (or lists) of corresponding x- and y-values,
       specifying the points in the x-y plane.  The x-values
       must be in increasing order.

     Output:
       S is a function that takes x or an array (or list) of x-values
         It evaluates the cubic spline and returns the interpolated value.

     Implementation:
       Note that there is one more "a" than "b" or "c". I would suggest
       the following mapping:

         a[0] = a_0           b[0] = b_1        c[0] = c_1
         a[1] = a_1           b[1] = b_2        c[1] = c_2
              :                   :                 :
         a[n-2] = a_(n-2)     b[n-2] = b_(n-1)  c[n-2] = c_(n-1)
         a[n-1] = a_(n-1)

       The polynomial piece is evaluated at xx using

         p_i(xx) = a[i]*(x[i+1]-xx)**3/(6*hi) + a[i+1]*(xx-x[i])**3/(6*hi) +
                   b[i]*(x[i+1]-xx) + c[i]*(xx-x[i])

       where hk = x[k+1] - x[k] for k = 0, ... , n-1
```

```
    '''
    n = len(x)
    h = np.zeros(n-1)
    a = np.zeros(n)
    b = np.zeros(n-1)
    c = np.zeros(n-1)

    M = np.zeros((n,n))
    r = np.zeros(n)

    # === YOUR CODE HERE ===

    # make h
    for i in range(0,n-1):
        h[i] = x[i+1] - x[i]
    # make r
    for i in range(1,n-1):
        r[i] = (y[i+1] - y[i]) / h[i] - (y[i] - y[i-1]) / h[i-1]
    # make M
    for m in range(1,n-1):
        M[m][m-1] = h[m-1] / 6
        M[m][m] = (h[m-1] + h[m]) / 3
        M[m][m+1] = h[m] / 6
    M[0][0] = 1
    M[n-1][n-1] = 1

    # make a
    a = np.linalg.solve(M,r)

    # make b and c
    for i in range(0,n-1):
        b[i] = y[i] / h[i] - a[i] * h[i] / 6
        c[i] = y[i+1] / h[i] - a[i+1] * h[i] / 6


    #=======================================
    #
    # This is the function that gets returned.
    # It evaluates the cubic spline at xvals.
    #
    def spline(xvals, x=x, a=a, b=b, c=c):
        '''
         S = spline(xvals)

         Evaluates the cubic spline at xvals.

         Inputs:
```

```
          xvals can be list-like, or a scalar (**must be in ascending order**)

         Output:
          S is a list of values with the same number of elements as x
        '''
        # Turn non-list-like input into list-like
        if type(xvals) not in (list, np.ndarray,):
            xvals = [xvals]

        S = []  # The return list of values

        #
        k = 0   # this is the current polynomial piece
        hk = x[k+1] - x[k]

        for xx in xvals:

            # If the next x-value is not on the current piece...
            if xx>x[k+1]:
                # ... Go to next piece
                k += 1
                hk = x[k+1] - x[k]

            S_of_x = a[k]*(x[k+1]-xx)**3/(6*hk) + a[k+1]*(xx-x[k])**3/(6*hk) +␣
↪b[k]*(x[k+1]-xx) + c[k]*(xx-x[k])

            S.append(S_of_x)

        return S
    #=======================================

    return spline
```

1.2 Test MySpline

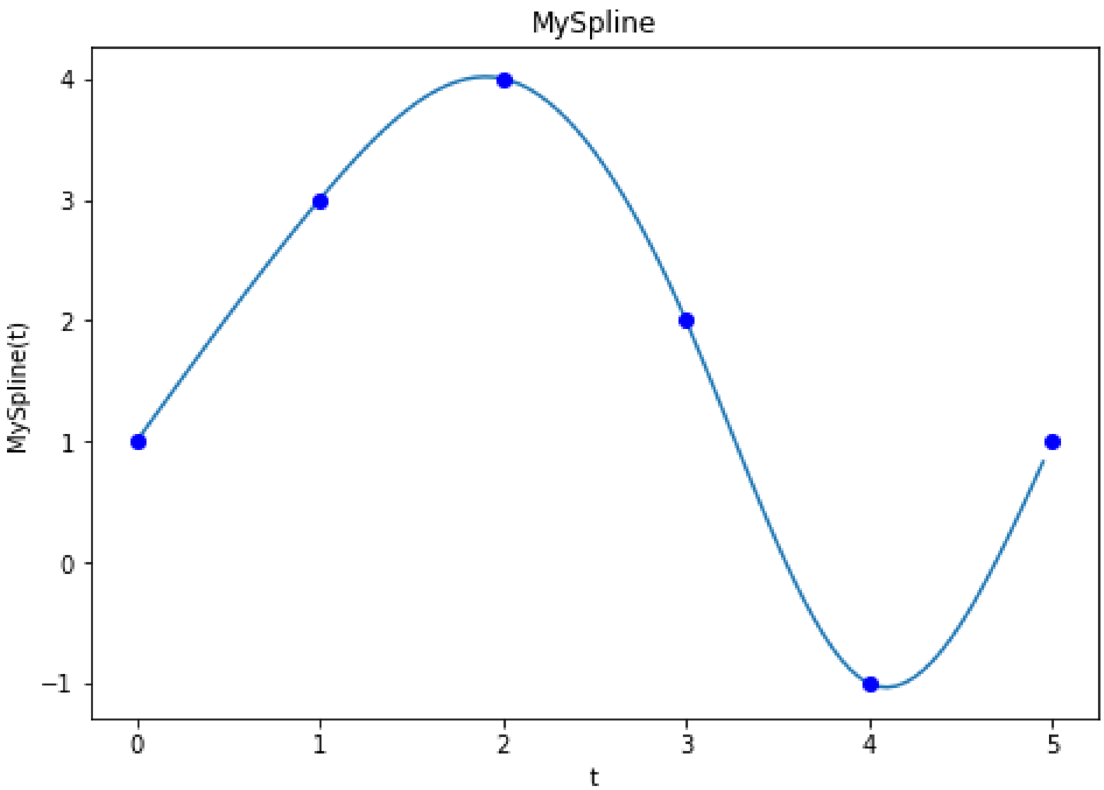
[3]:
```
# Simple data points to interpolate
y = [1, 3, 4, 2, -1, 1]
t = [0, 1, 2, 3,  4, 5]
```

[8]:
```
# Call the function
sp = MySpline(t,y)
```

[11]:
```
# Plot the spline and the interpolation points
xx = np.arange(0, 5, 0.05)
fig = plt.figure()
ax=fig.add_axes([0,0,1,1])
```

```
ax.plot(xx,sp(xx))
ax.plot(t,y,'bo')
ax.set_title("MySpline")
ax.set_xlabel('t')
ax.set_ylabel('MySpline(t)')
```

[11]: Text(0, 0.5, 'MySpline(t)')



[ ]: