Implementing Cubic Spline Interpolation

Paul Simmerling, *Electrical Engineering & Physics*, University of Connecticut

***Abstract*—The cubic spline interpolation method is discussed then derived. The interpolation is then implemented using Python3 and NumPy with a brief demonstration on several example data sets.**

I. Introduction

INTERPOLATION is a common method to make smooth curves using discrete points of data. Applications for interpolation are numerous, some applications include root finding, approximating derivatives and integrals, to even making smooth curves for graphic and computer-aided design.

Within this paper, the cubic spline will be investigated. First, the equations for the spline will be derived. Then the equations will be implemented using Python and NumPy. The code will be tested on various data sets to demonstrate its versatility.

II. Derivation

The cubic spline is a preferred method of interpolating data points as it is less likely to oscillate through data points. The spline is a piecewise function with n components connecting $n+1$ data points. Thus our functions can be denoted as $f_{0,1}(x), f_{1,2}(x), \ldots, f_{n,n+1}(x)$ where $f_{i,i+1}(x)$ connects the points i and $i+1$. Requiring continuity in the second derivative at each connecting data points, a boundary condition can be found.

$$f''_{i-1,i}(x_i) = f''_{i,i+1}(x_i) = k_i \tag{1}$$

Another boundary condition can be set, $k_0 = k_n = 0$. To find the coefficients, the linearity of the second derivative can be used in addition to Lagrange's two-point interpolation.

$$f''_{i,i+1}(x) = k_i l_i(x) + k_{i+1} l_{i+1}(x) \tag{2}$$

$$l_i \equiv \frac{x - x_{i+1}}{x_i - x_{i+1}} \tag{3}$$

$$l_{i+1} \equiv \frac{x - x_i}{x_{i+1} - x_i} \tag{4}$$

$$\begin{aligned} f''_{i,i+1}(x) &= k_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + k_{i+1} \frac{x - x_i}{x_{i+1} - x_i} \\ &= \frac{k_i(x - x_{i+1}) - k_{i+1}(x - x_i)}{x_i - x_{i+1}} \end{aligned} \tag{5}$$

Integrate eq. 5 twice to find $f_{i,i+1}(x)$

$$\begin{aligned} f_{i,i+1}(x) = &\frac{k_i(x - x_{i+1})^3 - k_{i+1}(x - x_i)^3}{6(x_i - x_{i+1})} \\ &+ A(x - x_{i+1}) - B(x - x_i) \end{aligned} \tag{6}$$

Imposing boundary conditions $f_{i,i+1}(x_i) = y_i$

$$y_i = \frac{k_i(x_i - x_{i+1})^3}{6(x_i - x_{i+1})} + A(x_i - x_{i+1}) \tag{7}$$

Solving for A

$$A = \frac{y_i}{x_i - x_{i+1}} - \frac{k_i}{6}(x_i - x_{i+1}) \tag{8}$$

Imposing boundary conditions $f_{i,i+1}(x_{i+1}) = y_{i+1}$

$$y_{i+1} = \frac{-k_{i+1}(x_{i+1} - x_i)^3}{6(x_i - x_{i+1})} - B(x_{i+1} - x_i) \tag{9}$$

Solving for B

$$B = \frac{y_{i+1}}{x_i - x_{i+1}} - \frac{k_{i+1}}{6}(x_i - x_{i+1}) \tag{10}$$

Substituting B and A back into eq. 6

$$\begin{aligned} f_{i,i+1}(x) = &\\ &\frac{k_i}{6}\left[\frac{(x - x_{i+1})^3}{x_i - x_{i+1}} - (x - x_{i+1})(x_i - x_{i+1})\right] \\ &- \frac{k_{i+1}}{6}\left[\frac{(x - x_i)^3}{x_i - x_{i+1}} - (x - x_{i+1})(x_i - x_{i+1})\right] \\ &+ \frac{y_i(x - x_{i+1}) - y_{i+1}(x - x_i)}{x_i - x_{i+1}} \end{aligned} \tag{11}$$

Using the continuity in the second derivative at $f''_{i-1,i}(x_i) = f''_{i,i+1}(x_i)$

$$\begin{aligned} &k_{i-1}(x_{i-1} - x_i) + 2k_i(x_{i-1} - x_{i+1}) + k_{i+1}(x_i - x_{i+1}) \\ &= 6\left(\frac{y_{i-1} - y_i}{x_{i-1} - x_i} - \frac{y_i - y_{i+1}}{x_i - x_{i+1}}\right), i = 1, 2, \ldots, n-1 \end{aligned} \tag{12}$$

This equation can be solved using LU decomposition as it is a tridiagonal coefficient matrix. Now that the cubic spline coefficients can be calculated, the equations can be implemented in python.

III. Python Implementation

For the implementation, Python3 will be used in addition to NumPy for handling the arrays. The first task is to use eq. 12 to determine k_i for the spline.

```
def curvatures(xData, yData):
  n = len(xData) - 1
  #Create Tridiag matrix with d on
  #main diag
  #c on upper diag, e on lower
  #diag and k as B
  #see eq 12
  c, d = np.zeros(n), np.ones(n+1)
  e, k = np.zeros(n),\
         np.zeros(n+1)
  c[0:n-1] = xData[0:n-1] - \
             xData[1:n]
  d[1:n] = 2.0*(xData[0:n-1] -\
           xData[2:n+1])
  e[1:n] = xData[1:n] -\
          xData[2:n+1]
  k[1:n] = 6.0*(yData[0:n-1] -\
           yData[1:n]) \
            /(xData[0:n-1] -\
            xData[1:n])  \
          -6.0*(yData[1:n] -\
          yData[2:n+1]) \
            /(xData[1:n] -\
            xData[2:n+1])
  #Decompose tridiag and solve
  LUdecomp3(c,d,e)
  LUsolve3(c,d,e,k)
  return k
```

In this code, four vectors are created that represent the diagonals of the tridiagonal matrix and the solution vector. Then the matrix is solved using standard LUdecomposition techniques.

Now, the spline must be evaluated at x using eq. 11

```
def evalSpline(xData,yData,k,x):
  #Determine which f_{i,i+1}(x) is
  #used
  def findSegment(xData,x):
    iLeft = 0
    iRight = len(xData)-1
    while True:#Binary
    #bisection
      #If stuck between 2 f,
      #use left f
      if (iRight-iLeft) <= 1:
        return iLeft
      #choose middle data point
      i = int((iRight-iLeft)/2\
        +iLeft)
      #check to see if x is
      #less than data point
      if x < xData[i]: iRight = i
      else: iLeft = i

  #Find which f
  i = findSegment(xData,x)
  #Make math easier to read
  h = xData[i] - xData[i+1]
  #Use equation 11 to
  #evaluate spline
  y = ((x-xData[i+1])**3/h - (x -\
    xData[i+1])*h)*k[i]/6.0 \
      -((x-xData[i])**3/h - \
      (x-xData[i])*h)*k[i+1]/6.0\
      + (yData[i]*(x-xData[i+1]) \
      - yData[i+1]*(x-xData[i]))/h
  return y
```

For evaluating the spline, the exact $f_{i,i+1}(x)$ must be determined. This is done with a binary search to find where x lies in the data $x_i = X$. Then the cubic spline can be evaluated using eq. 11 and the appropriate k_i.

IV. Tests

It is important to test the implementation on some example data.

The first set of data is a simple oscillator such as that from a sine wave. This data can be seen in table 1 Using the code provided, the k coefficients are

X	1	2	3	4	5
Y	0	1	0	1	0

TABLE I
Data for the first test

are determined and provided in table 2. Plotting the spline, provides the behavior seen in fig 1. From this plot, the oscillating behavior is shown to only exist

k_0	k_1	k_2	k_3	k_4
0	-4.286	5.143	-4.286	5

TABLE II
k CUBIC SPLINE COEFFICIENTS FOR THE DATA IN TABLE 1.

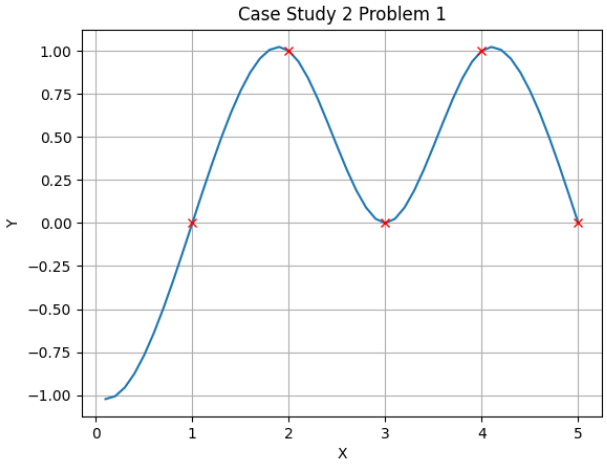

Fig. 1. Plot of cubic spline for the data for test 1.

in the range data range. It does not extrapolate the oscillating outside of the provided range. Another interesting behavior is the repetition of y values for different x values. Using the test range of 0.1 to 5 with increments of 0.1, we find the following sub-sample of shared y values (table 3). A more complete list is provided in the appendix.

x	y
5.0	0.0
1.0	0.0
3.0	0.0
2.9	0.024143
3.1	0.024143
2.8	0.090286
3.2	0.090286
4.9	0.170714
1.1	0.170714

TABLE III
SHARED Y-VALUES USING THE CUBIC-SPLINE FOR TEST 1.

Another set of data that will provide a good test of the code is only three data points. This can be see in table 4, table 5, and fig. 2.

x	0	1	2
y	0	2	1

TABLE IV
DATA FOR TEST 2.

For this data, an interesting behavior can be seen. If $x = i = 1$ is evaluated, the resultant value of $y = j = 2$ is found. If $x = j = 2$ is evaluated, the resultant value of $y = i = 1$ is found.

k_0	k_1	k_2
0	-4.5	0

TABLE V
CUBIC SPLINE COEFFICIENTS, k, FOR TEST 2.

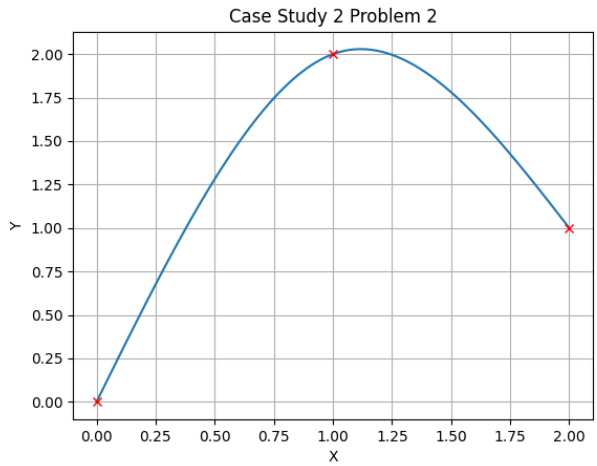

Fig. 2. Cubic Spline for Test 2.

A third test can be performed with the data shown in table 5. The k coefficients are in table 6, and a plot is seen in fig 3.

x	1	2	3	4	5
y	13	15	12	9	13

TABLE VI
DATA FOR TEST 3.

k_0	k_1	k_2	k_3	k_4
0	-7.285714	-0.857143	10.714286	0

TABLE VII
CUBIC SPLINE COEFFICIENTS, k, FOR TEST 3.

Evaluating this spline at $x = 3.4$, the value of $y = 10.254857$ is found.

A final data set can be tested. However, the cubic spline interpolant of $X = Y(x)$ will be determined. This involves sorting the data to be in ascending order of for Y. The data can be see in table 7, coefficients in table 8. A plot of the data is seen in figure 4.

The roots of this cubic spline can be found at $y = 1.445000$ and $y = 2.111400$. However, it is important to note that attempting to evaluate the function outside of the provided data range will result in strange behaviors. This can be shown in

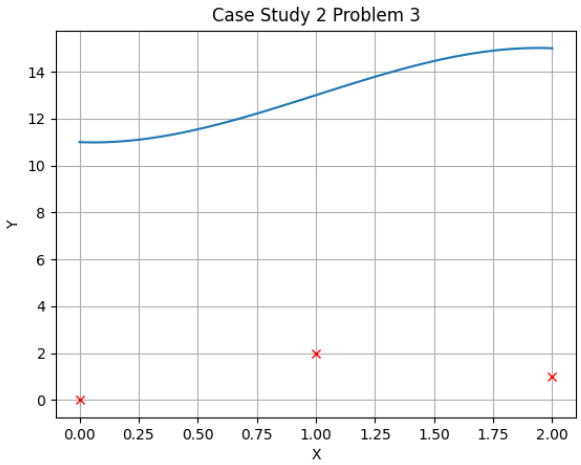

Fig. 3. Cubic Spline for Test 3.

x	0.2	0.4	0.6	0.8	1.0
y	1.150	0.855	0.377	-0.266	-1.049

TABLE VIII
DATA FOR TEST 4.

figure 5. As the y values are increased, the predicted x values exponentially increase.

V. CONCLUSION

Cubic spline interpolation is an important tool in any numerical analysis toolbox. With applications from approximating integrals and derivatives to forming smooth lines for modeling objects, it is very useful. Implementing it in python is relatively simple and can be done to produce efficient results. However, it is important to note that cubic spline is meant for interpolating data and can produce erroneous results if it is used to extrapolate the behavior of the data.

k_0	k_1	k_2	k_3	k_4
0	-0.106883	-0.04487493	-0.99345805	0

TABLE IX
CUBIC SPLINE COEFFICIENTS, k, FOR TEST 4.

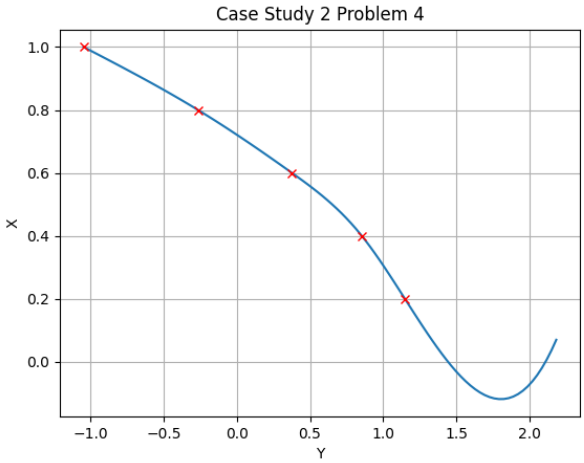

Fig. 4. Cubic Spline for Test 4.

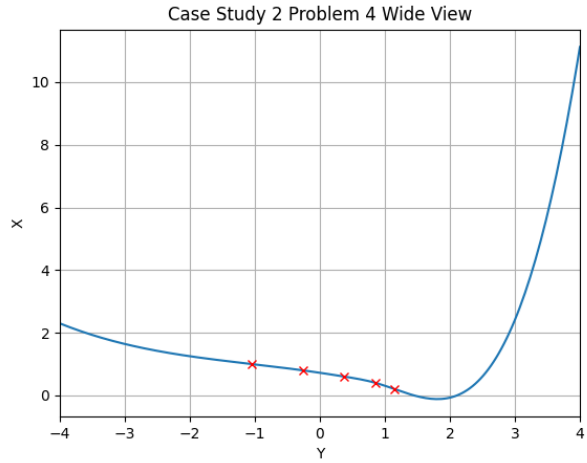

Fig. 5. Cubic Spline for Test 4 with large y.

APPENDIX
SHARED Y-VALUES FOR TEST 1

x	y
5.0	-0.0
1.0	0.0
3.0	0.0
2.9	0.024143
3.1	0.024143
2.8	0.090286
3.2	0.090286
4.9	0.170714
1.1	0.170714
2.7	0.189
3.3	0.189
2.6	0.310857
3.4	0.310857
4.8	0.337143
1.2	0.337143
2.5	0.446429
3.5	0.446429
4.7	0.495
1.3	0.495
2.4	0.586286
3.6	0.586286
1.4	0.64
4.6	0.64
2.3	0.721
3.7	0.721
4.5	0.767857
1.5	0.767857
2.2	0.841143
3.8	0.841143
1.6	0.874286
4.4	0.874286
2.1	0.937286
3.9	0.937286
4.3	0.955
1.7	0.955
4.0	1.0
2.0	1.0
4.2	1.005714
1.8	1.005714
4.1	1.022143
1.9	1.022143

TABLE X
SHARED Y WITH TOLERANCE OF 6 SIGNIFICANT FIGURES

APPENDIX
CODE FOR PYTHON3 CUBIC SPLINE

```
#Case Study 2
from math import *
import numpy as np

def LUdecomp3(c,d,e):
  n = len(d)
  for k in range(1,n):
    lam = c[k-1]/d[k-1]
```

```
    d[k] -= lam*e[k-1]
    c[k-1] = lam
  return c,d,e
def LUsolve3(c,d,e,b):
  n = len(d)
  for k in range(1,n):
    b[k] -= c[k-1]*b[k-1]
  b[n-1] = b[n-1]/d[n-1]
  for k in range(n-2, -1, -1):
    b[k] = (b[k]-e[k]*b[k+1])/d[k]
  return b

def curvatures(xData, yData):
  n = len(xData) - 1
  #Create Tridiag matrix with d on main diag
  #c on upper diag, e on lower diag and k as B
  #see eq 12
  c, d = np.zeros(n), np.ones(n+1)
  e, k = np.zeros(n), np.zeros(n+1)
  c[0:n-1] = xData[0:n-1] - xData[1:n]
  d[1:n] = 2.0*(xData[0:n-1] - xData[2:n+1])
  e[1:n] = xData[1:n] - xData[2:n+1]
  k[1:n] = 6.0*(yData[0:n-1] - yData[1:n]) \
            /(xData[0:n-1] - xData[1:n])  \
          -6.0*(yData[1:n] - yData[2:n+1]) \
            /(xData[1:n] - xData[2:n+1])
  #Decompose tridiag and solve
  LUdecomp3(c,d,e)
  LUsolve3(c,d,e,k)
  return k

def evalSpline(xData,yData,k,x):
  #Determine which f_{i,i+1}(x) is used
  def findSegment(xData,x):
    iLeft = 0
    iRight = len(xData)-1
    while True:#Binary bisection
      #If stuck between 2 f, use left f
      if (iRight-iLeft) <= 1: return iLeft
      #choose middle data point
      i = int((iRight-iLeft)/2+iLeft)
      #check to see if x is less than data point
      if x < xData[i]: iRight = i
      else: iLeft = i

  #Find which f
  i = findSegment(xData,x)
  #Make math easier to read
  h = xData[i] - xData[i+1]
  #Use equation 11 to evaluate spline
```

```
  y = ((x-xData[i+1])**3/h - (x - xData[i+1])*h)*k[i]/6.0 \
      -((x-xData[i])**3/h - (x - xData[i])*h)*k[i+1]/6.0   \
      + (yData[i]*(x-xData[i+1]) \
      - yData[i+1]*(x-xData[i]))/h
  return y

import matplotlib.pyplot as plt
#Plotting spline and data helper function
def myPlot(X, Y, xs, ys, title, fname, xlabel='X', ylabel='Y'):
  plt.clf()
  plt.plot(xs,ys)#Plot spline
  plt.plot(X,Y,'rx')#Plot discrete data
  plt.grid()
  plt.xlabel(xlabel)
  plt.ylabel(ylabel)
  plt.title(title)
  plt.savefig("figures/"+fname)
  #plt.show()

print("+------------------+")
print("|   Question 1   +")
print("+------------------+")
X = np.array([1,2,3,4,5])
Y = np.array([0,1,0,1,0])
k = curvatures(X,Y)#Find spline
print(k)
print(evalSpline(X,Y,k,1.5))
#Make test x from 0.1 to 5 on .1 intervals
xs = np.arange(0.1, 5.1, 0.1)
ys = np.array([0]*len(xs),dtype=float)
for i in range(len(xs)):#Eval each x with spline
  ys[i] = evalSpline(X,Y,k,xs[i])
asort = ys.argsort()#Sort x and y based on y
xas = xs[asort]
yas = ys[asort]
for i in range(len(xas)):#iterate through x
  tol = 1e-6
  #if y is within tol of surrounding data, print x,y
  if i > 1 and abs(yas[i-1]-yas[i])<tol: print(xas[i], yas[i])
  elif i<(len(xas)-1) and abs(yas[i+1]-yas[i])<tol: print("\n",xas[i], yas[i])
#Make plot
myPlot(X,Y,xs,ys,'Case Study 2 Problem 1','cs2p1.png')

print("+------------------+")
print("|   Question 2   +")
print("+------------------+")
X = np.array([0,1,2.])
Y = np.array([0,2,1])
k = curvatures(X,Y)#find spline
print(k)
```

```
tol,di,found = 1e-6,0.01, False
while di > tol:#if not finding data, try smaller points
  for i in np.arange(0, 1+di, di):#keep 0<=i<=1
    j = evalSpline(X,Y,k,i)#eval i for j
    if j <= 2 and j >= 1:#if j in range 1<=j<=2
      i2 = evalSpline(X,Y,k,j)#eval j for i2
      if abs(i2-i)<tol: #if eval j is within tol of i
        print(i,j)
        found = True#don't need more data
        break
  if found or di < tol: break#Finish if found a point
                             # or too small
  else: di = di/10#If didn't find, try more data

#Make spline for plotting
xs = np.linspace(min(X),max(X),100)
ys = np.array([0]*len(xs),dtype=float)
for i in range(len(xs)):
  ys[i] = evalSpline(X,Y,k,xs[i])
myPlot(X,Y,xs,ys,'Case Study 2 Problem 2','cs2p2.png')

print("+------------------+")
print("|   Question 3   +")
print("+------------------+")
#Make function for easy eval
def evalQ3(x):
  X = np.array([ 1, 2, 3,4, 5.])
  Y = np.array([13,15,12,9,13.])
  k = curvatures(X,Y)
  return evalSpline(X,Y,k,x)

print(evalQ3(3.4))
xs = np.linspace(min(X),max(X),100)
ys = np.array([0]*len(xs),dtype=float)
for i in range(len(xs)):
  ys[i] = evalQ3(xs[i])
myPlot(X,Y,xs,ys,'Case Study 2 Problem 3','cs2p3.png')
#plt.show()



print("+------------------+")
print("|   Question 4   +")
print("+------------------+")
#FIND Y(X)
X = np.array([  0.2,  0.4,  0.6,   0.8,   1.0])
Y = np.array([1.150,0.855,0.377,-0.266,-1.049])
#Resort data so we can find spline Y(x)
asort = Y.argsort()
X = X[asort]
```

```
Y = Y[asort]
k = curvatures(Y,X)
print(k)

#Find zeros (roots) of Y(x)
from hw3 import multisearch_ridder
def f(y): return evalSpline(Y,X,k,y)
multisearch_ridder(f,1,3)

#Plot spline with roots
ys = np.linspace(min(Y),max(Y)*1.9,100)
xs = np.array([0]*len(ys),dtype=float)
for i in range(len(ys)):
  xs[i] = evalSpline(Y,X,k,ys[i])
myPlot(Y,X,ys,xs,'Case Study 2 Problem 4','cs2p4.png','Y','X')

#plot spline show behavoior is unpredicable outide of data range
ys2 = np.linspace(floor(min(Y)*3),ceil(max(Y)*3),200)
xs2 = np.array([0]*len(ys2),dtype=float)
for i in range(len(ys2)):
  xs2[i] = evalSpline(Y,X,k,ys2[i])
myPlot(Y,X,ys2,xs2,'Case Study 2 Problem 4 Wide View','cs2p4_wide.png','Y','X')
plt.xlim(min(ys2),max(ys2))
plt.savefig('figures/cs2p4_wide.png')
```