PETR 5313: CRN 38950, Fall 2017
Numerical Application in Petroleum Engineering, Lesson 08: Interpolation, Extrapolation, and Integration

Ekarit Panacharoensawad, PhD
Terry Fuller Petroleum Engineering Research Building Room 236
ekarit.panacharoensawad@ttu.edu

Cite as: Panacharoensawad, E. (2017) "Numerical application in petroleum engineering, Lesson 08: Interpolation - Extrapolation", presentation slide for Texas Tech PETR 5313 Fall 2017

Outline

- Interpolation
- Extrapolation
  - 1D data
  - 2D data
  - Linear
  - Cubic spline

Linear Interpolation

Draw a straight line connected two dots A and B to tell the f(x) value based on the give x value. $x \in [A, B]$

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$y = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0}$$

$$y = \frac{x_1 - x}{x_1 - x_0}y_0 + \frac{x - x_0}{x_1 - x_0}y_1$$

Method for interpolation

Given data: $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$ $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]$

Target: y value at x_t (`val`)

Manual

- Need to find x_i and x_{i+1} that x_t stays in between
  - `idx = np.abs(x-val).argmin()`
    - `x` is list, `.argmin()` find the index where minimum value occur
  - `idx2 = np.searchsorted(x, val) – 1`
    - This give the index on the left boundary

Method for interpolation

Automatic Function: `numpy.interp`

`np.interp(val,x,y)`

- This is to get y at x value equal to `val`

`val` can be list of values too

- This does interpolation several times

Linear Extrapolation

Extend a straight line outside the domain to get the value of y at an interested x

- Equation is the same as linear equation

Manual method

- Need to find if extrapolation is on left or right side of the domain

Automatic

- `from scipy import interpolate`
- `Lin_inter = interpolate.interp1d(x,y,kind='linear',fill_value='extrapolate')`
- `y_sp = Lin_inter(val)`

Interpolation for 2D data: Use Scipy

```
model = interpolate.interp2d(x = T_in,
                             y = P_in, z = Mu_in)
val = (70,3000)
z_pred = model(*val)
```

P_in.shape gives (11,)

T_in.shape gives (7,)

Mu_in.shape gives (11, 7)

Interpolation for 2D data: Manual

From (z,T) = (109.7, 66.7) and (93.5, 75) at P = 2550

- Find z at T = 70.0 (target) by linear interpolation

From (z,T) = (97.5, 66.7) and (81.2, 75) at P = 3040

- Find z at T = 70.0 (target) by linear interpolation

From (z,P) = (103.2, 2550) and (90.978, 3040) at T = 70

- Find z at P = 3000 (target) by linear interpolation

$$\begin{bmatrix} & \text{T\_in}[2] = 66.7 & \text{x\_target} = 70.0 & \text{T\_in}[3] = 75.0 \\ \text{P\_in}[5] = 2550.0 & 109.7 & 103.2 & 93.5 \\ \text{y\_target} = 3000 & & 91.9776 & \\ \text{P\_in}[6] = 3040.0 & 97.5 & 90.978 & 81.2 \end{bmatrix}$$

Simple Cubic Spline

Input: Slope + coordinate of 2 points

- (x1, y1), (x2, y2)
- Slope at (x1, y1) and (x2, y2)

Output:

- Curve that pass those 2 points and have the specified slope

Simple Cubic Spline: Curve fitting equation

q is the cubic spline prediction

k is the slope

$$q = (1-t)y_1 + ty_2 + t(1-t)(a(1-t) + bt)$$

$$t = \frac{x - x_1}{x_2 - x_1}$$

$$a = k_1(x_2 - x_1) - (y_2 - y_1)$$

$$b = -k_2(x_2 - x_1) + (y_2 - y_1)$$

Simple Cubic Spline: Example

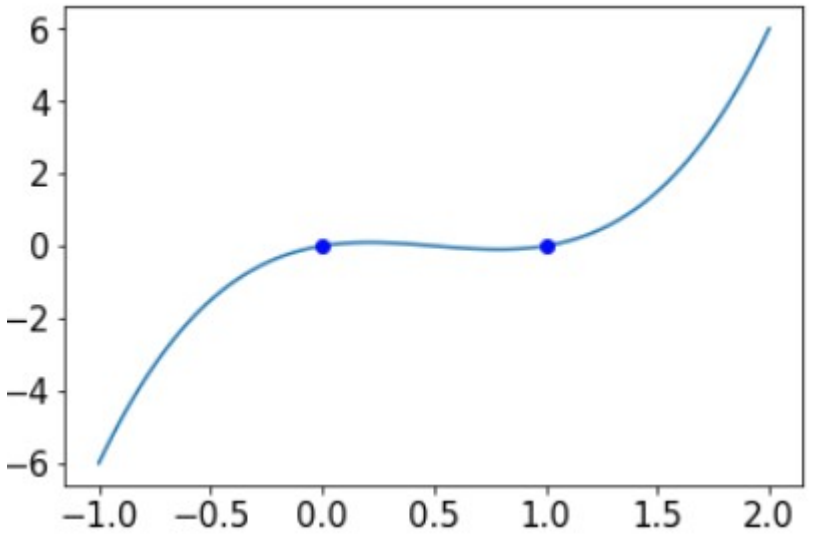
For $x1 = 0$ $x2 = 1$

$y1 = 0$ $y2 = 0$

$f'(x1) = 1$ $f'(x2) = 1$

Find the interpolation / extrapolation curve from cubic spline

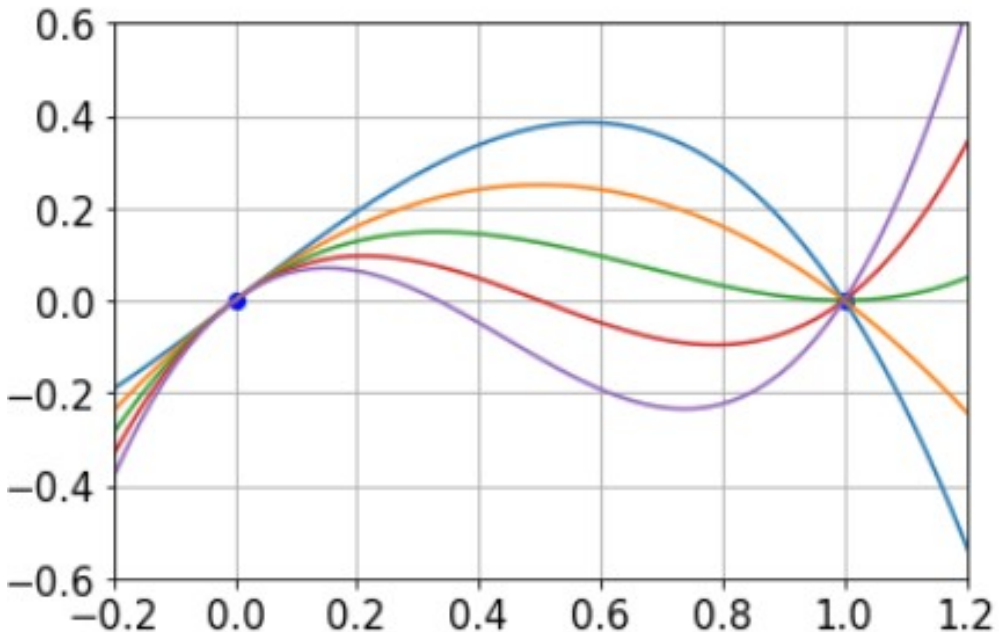
```
x = np.linspace(-1,2,100)
t = (x-x1)/(x2-x1)
a = k1*(x2 - x1) - (y2 - y1)
b = -k2*(x2 - x1) + (y2 - y1)
q = (1 - t)*y1 + t*y2 + t*(1 - t)*(a*(1 - t) + b*t)
```



Simple Cubic Spline for Various Cases

For k2 in [-2, -1, 0, 1, 2], we have

Cubic spline adjusts the interpolation curve according to the specified slope



Cubic Spline for Multiple Points

Advantages of using cubic spline

- Cubic spline can approximate the slope between points accurately (in addition to the function value)

Use Cubic spline when trying to approximate the slope of function based on the samples of function values

Output properties

- q(x), q’(x), and q’’(x) are piecewise continuous functions!

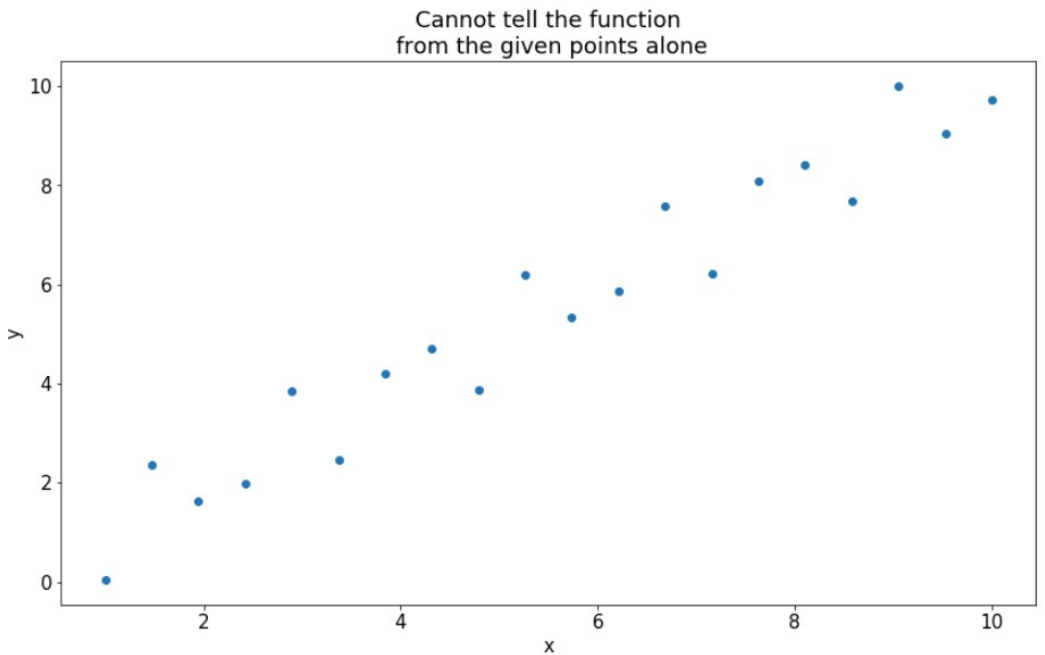
Cubic Spline for Multiple Points: Natural B.C.

- When we have many points, we typically do not know the slope at each point.
- To use cubic spline without knowing the slope, the boundary condition on each end should be specified
- One simple and powerful assumption is that q''(x0) and q''(xn) = 0
  - This is called "natural boundary condition"
  - This mean that the curves are just straight at both ends location

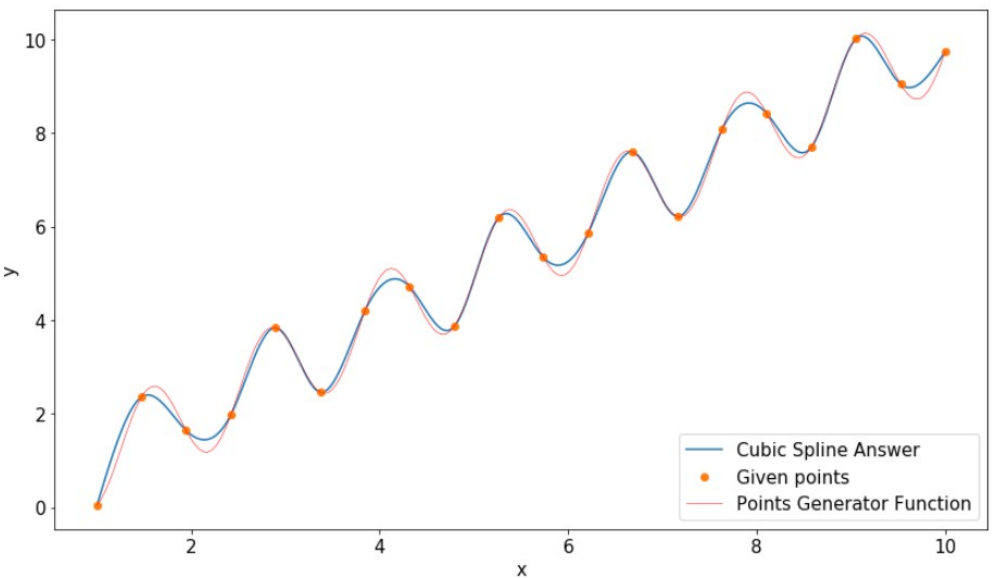
Cubic Spline for Multiple Points: Example

Cubic Spline solution was done by not knowing the function used to generate the data points

Before cubic spline interpolation

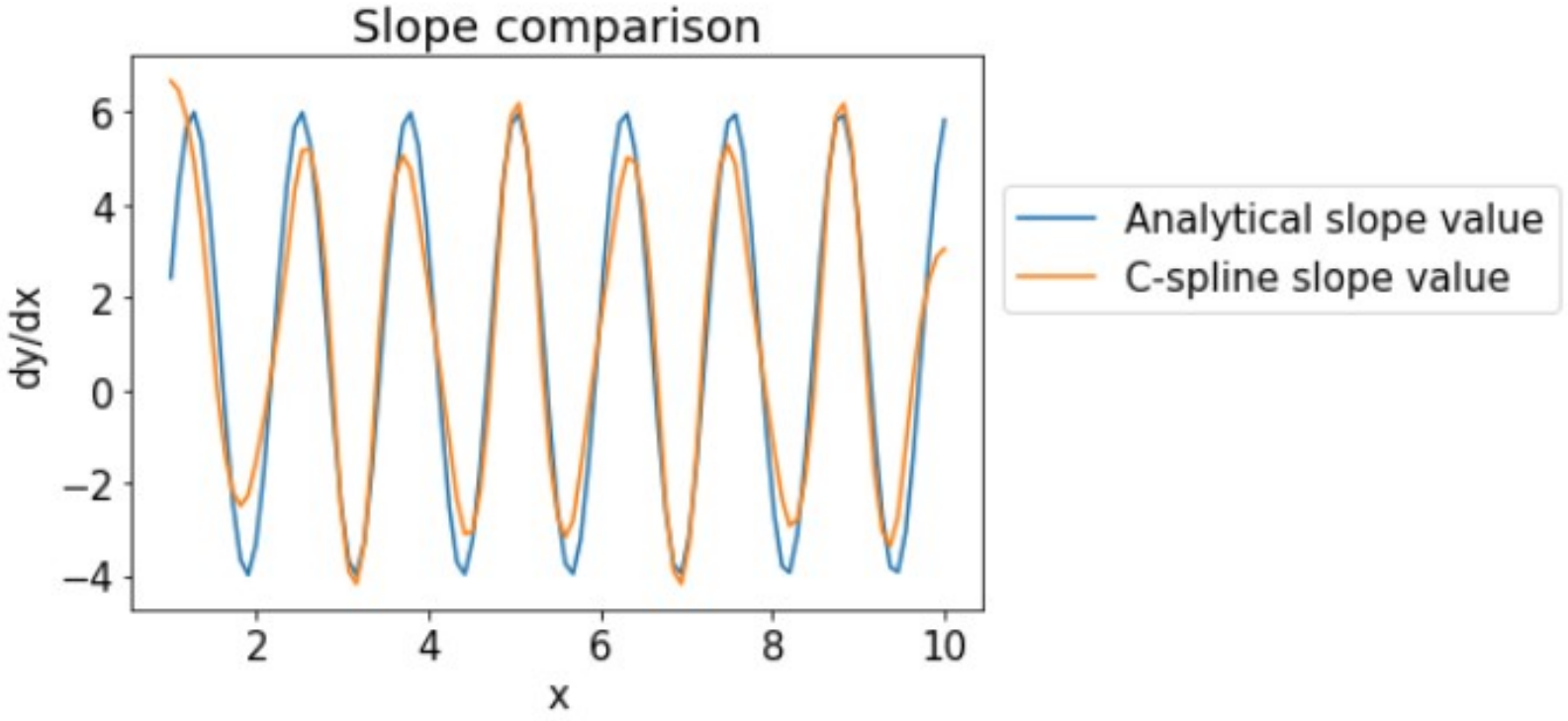


After cubic spline interpolation



Cubic Spline for Multiple Points: Example

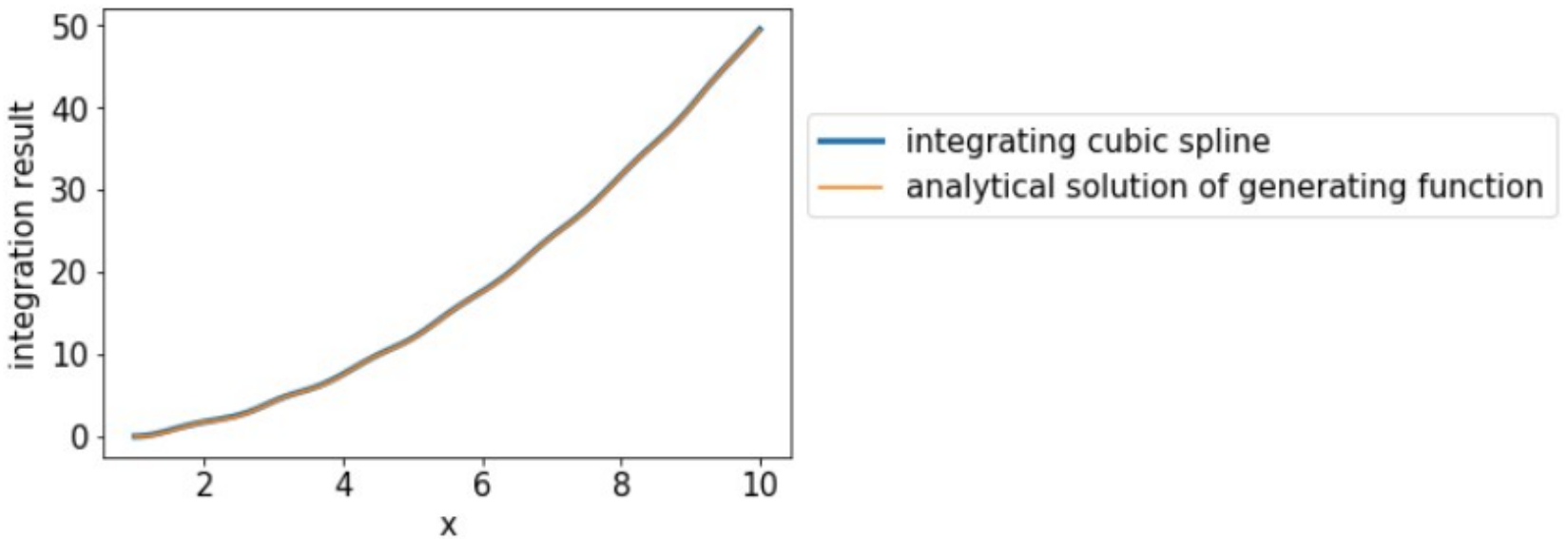
Slope from cubic spline is comparable to the analytical solution



Cubic Spline for Multiple Points: Example

Integration of the approximation function is also close to the analytical solution of the original equation

$$\int_1^x f(x)dx$$



Cubic Spline for Multiple Points: Calculation Steps

- Solve for slope at each points by assume the second derivative to be zero for the end points (natural B.C.)
- Once the slopes are obtained, use Simple cubic spline approach to get the interpolation value in each interval

Variable definition

- k = slope (to be calculated)
- x[i] = given x-coordinate of the point i
- y[i] = given y-coordinate of the point i

Cubic Spline: Slope Calculation

- Solve system of linear equations to get all k

$$B_0 k_0 + C_0 k_1 = D_0 \qquad \text{for i = 0}$$

$$A_i k_{i-1} + B_i k_i + C_i k_{i+1} = D_i \qquad \text{for i = 1 to n-1}$$

$$A_n k_{n-1} + B_n k_n = D_n \qquad \text{for i = n}$$

$$\begin{bmatrix} B_0 & C_0 & & & & \\ A_1 & B_1 & C_1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & A_{n-1} & B_{n-1} & C_{n-1} \\ & & & & A_n & B_n \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ \vdots \\ k_{n-1} \\ k_n \end{bmatrix} = \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ D_{n-1} \\ D_n \end{bmatrix}$$

Cubic Spline: Slope Calculation...

Derivation: https://en.wikipedia.org/wiki/Spline_interpolation

$$2(x_1 - x_0)\,k_0 + (x_1 - x_0)\,k_1 = 3(y_1 - y_0)$$

$$\frac{k_{i-1}}{x_i - x_{i-1}} + \left(\frac{1}{x_i - x_{i-1}} + \frac{1}{x_{i+1} - x_i}\right) 2k_i + \frac{k_{i+1}}{x_{i+1} - x_i} =$$

$$3\left(\frac{y_i - y_{i-1}}{(x_i - x_{i-1})^2} + \frac{y_{i+1} - y_i}{(x_{i+1} - x_i)^2}\right)$$

$$(x_n - x_{n-1})\,k_{n-1} + 2(x_n - x_{n-1})\,k_n = 3(y_n - y_{n-1})$$

Cubic Spline: Slope Calculation...

Coefficient for system of linear equation

$$A_i = \frac{1}{x_i - x_{i-1}}$$

$$B_i = 2\left(\frac{1}{x_i - x_{i-1}} + \frac{1}{x_{i+1} - x_i}\right)$$

$$C_i = \frac{1}{x_{i+1} - x_i}$$

$$D_i = 3\left(\frac{y_i - y_{i-1}}{(x_i - x_{i-1})^2} + \frac{y_{i+1} - y_i}{(x_{i+1} - x_i)^2}\right)$$

Cubic Spline: Solving Manually

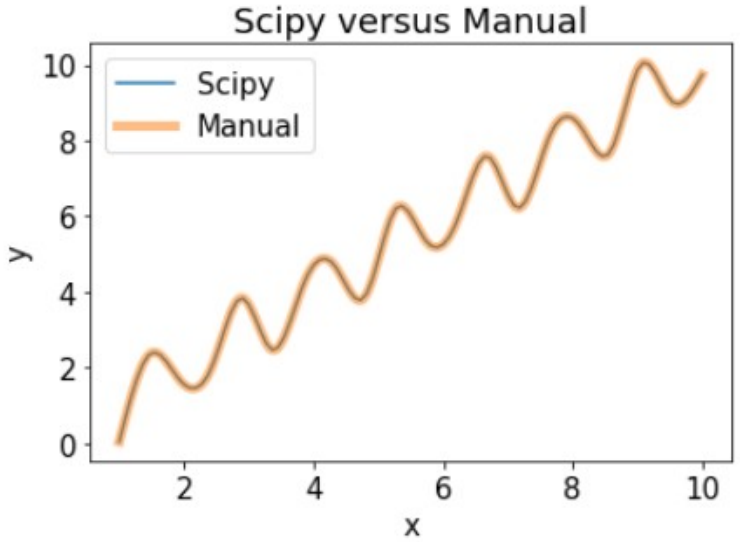
```
B0 = 2*(x[1]-x[0])
C0 = x[1]-x[0]
D0 = 3*(y[1]-y[0])
Ai = [1.0/(x[i]-x[i-1]) for i in range(1,len(x)-1)]
Bi = [2 * (1.0/(x[i]-x[i-1]) + 1/(x[i+1]-x[i])) for i in range(1,len(x)-1)]
Ci = [1.0/(x[i+1]-x[i]) for i in range(1,len(x)-1)]
Di = [3*((y[i]-y[i-1])/(x[i]-x[i-1])**2 + (y[i+1]-y[i])/(x[i+1]-x[i])**2)
      for i in range(1,len(x)-1)]
n = len(x) - 1 # the index is from 0 to n-1, not to n
An = x[n]-x[n-1]
Bn = 2*(x[n]-x[n-1])
Dn = 3*(y[n]-y[n-1])
AA = Ai+[An] # + operation for list is concatenation (similar to .append)
BB = [B0] + Bi + [Bn]
CC = [C0] + Ci
DD = [D0] + Di + [Dn]
CSpline = sp.sparse.diags(diagonals=[BB,AA,CC], offsets=[0,-1,1])
```

Cubic Spline: Using automatic function

model = sp.interpolate.CubicSpline(x,y,bc_type='natural')

y_pred = model(np.linspace(1,10,100))

Scipy automatic function is shorter and it is doing the same things with the manual implementation



Calculate Slope / Integrating Area from Cubic Spline

Get prediction of y, y', and y''

- y_pred = model(np.linspace(1,10,1000))
- y_slope_pred = model(np.linspace(1,10,1000),nu=1)
- y_slope2_pred = model(np.linspace(1,10,1000),nu=2)

Get integration result

- x_inte = np.linspace(1,10,101)
- y_inte = [model.integrate(1,i) for i in x_inte]