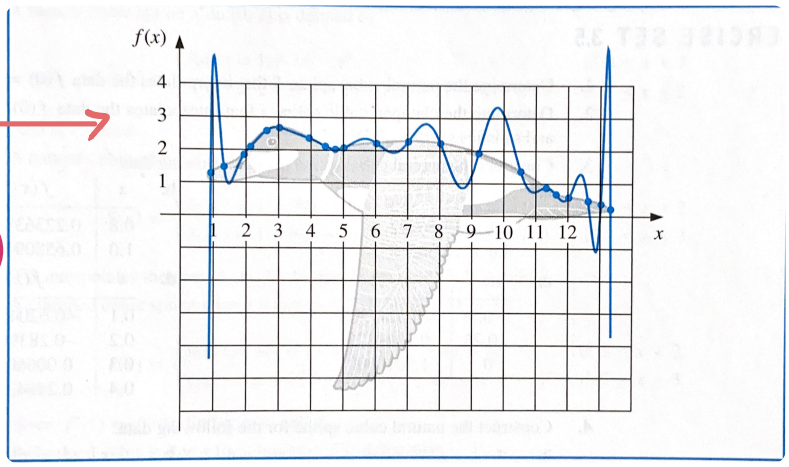
Cubic Spline Interpolation

Goal: So far we have studied several ways to construct an approximation for an arbitrary function over some interval using a single polynomial.



Problem: High-degree polynomials can oscillate erratically.

Solution:

- Divide the approximation interval into a collection of subintervals.
- Construct a generally different approximating

polynomial on each subinterval.

↳ Piecewise - polynomial approximation

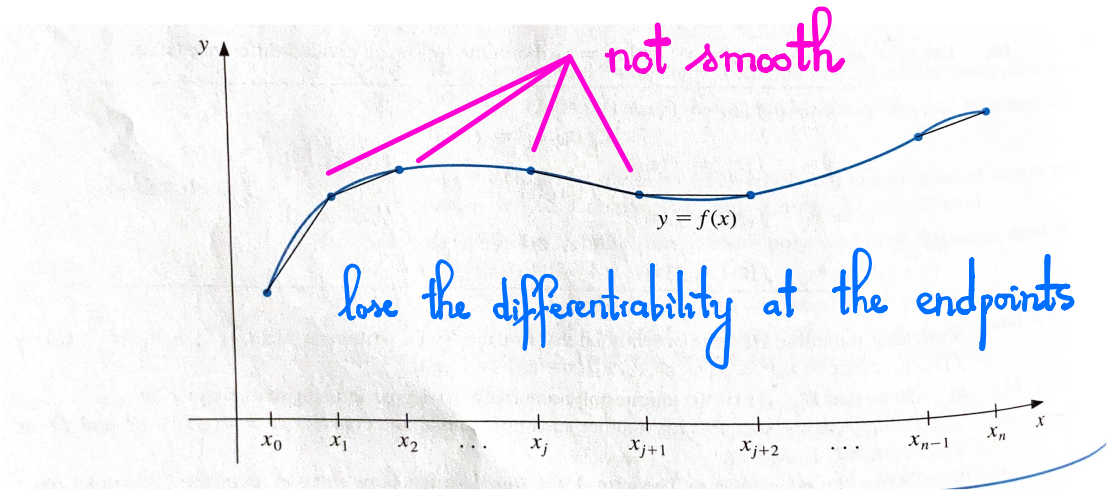
Piecewise - polynomial approximation

The simplest case : piecewise - linear interpolation

→ connect a set of data points

$(x_0, f(x_0))$, $(x_1, f(x_1))$, ..., $(x_n, f(x_n))$

by a series of straight lines.



Drawback the interpolating function is not smooth.

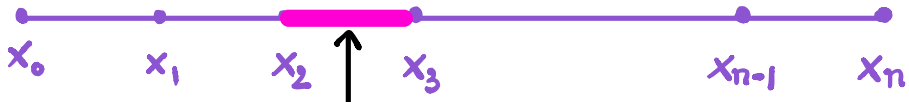
Physical conditions $\xrightarrow{\text{require}}$ "smoothness"

$\downarrow$

the approximating function should be continuously differentiable

Idea: Use a piecewise polynomial of Hermite type.

f & f' are known at these points



Construct a cubic Hermite polynomial on each subinterval $[x_i, x_{i+1}]$

get a function that has a continuous derivative on the interval $[x_0, x_n]$

• The cubic Hermite polynomial on $[x_i, x_{i+1}]$

$= H_3(x)$

$= f(x_i) H_{1,i}(x) + f'(x_i) \hat{H}_{1,i}(x) + f(x_{i+1}) H_{1,i+1}(x)$

$+ f'(x_{i+1}) \hat{H}_{1,i+1}(x)$

$$\begin{cases} H_{1,j}(x) = [1 - 2(x - x_j) L'_{1,j}(x_j)] L^2_{1,j}(x) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j = i, i+1 \\ \hat{H}_{1,j}(x) = (x - x_j) L^2_{1,j}(x) \end{cases}$$

$\longrightarrow$ we need to know $f'(x_i)$ ← unavailable

◎ Goal: consider approximation using piecewise polynomials that require no specific derivative info, except perhaps at $[x_0, x_n]$.

$\longrightarrow$ The simplest type of differentiable piecewise-polynomial function on $[x_0, x_n]$ is $P(x)$

$$P(x) = \begin{cases} S_0(x): \text{quadratic on } [x_0, x_1] \\ S_1(x): \text{quadratic on } [x_1, x_2] \\ \vdots \\ S_j(x): \text{quadratic on } [x_j, x_{j+1}] \\ \vdots \\ S_{n-1}(x): \text{quadratic on } [x_{n-1}, x_n] \end{cases}$$

$S_0(x_0) = f(x_0)$ $\quad$ $S_0(x_1) = f(x_1)$

$S_1(x_1) = f(x_1)$ $\quad$ $S_1(x_2) = f(x_2)$

$\vdots$

$S_j(x_j) = f(x_j)$ $\quad$ $S_j(x_{j+1}) = f(x_{j+1})$

$\vdots$

$S_{n-2}(x_{n-2}) = f(x_{n-2})$ $\quad$ $S_{n-2}(x_{n-1}) = f(x_{n-1})$

$S_{n-1}(x_{n-1}) = f(x_{n-1})$ $\quad$ $S_{n-1}(x_n) = f(x_n)$

$$S_j(x) = a_j x^2 + b_j x + c_j$$

Use conditions: $S_j(x_j) = f(x_j)$ and $S_j(x_{j+1}) = f(x_{j+1})$

⟶ we have the flexibility to choose S_j's so that the interpolant $P(x)$ has a continuous derivative on $[x_0, x_n]$.

⟶ The difficulty : need to specify conditions about $P'(x)$ at the endpoints x_0 and x_n.

There is not a sufficient # of constants to ensure that the conditions will be satisfied.

Cubic Splines

- Use cubic polynomials between each successive pair of nodes ⟶ the most common piecewise-polynomial approximation.
- This is called cubic spline interpolation.

⟶ There's enough flexibility to ensure the interpolant $P(x)$

- P is continously differentiable
- $P''(x)$ is continuous

• The construction of the cubic spline does not assume that $P'(x_i) = f'(x_i)$.

Definition:

• Given f on $[a,b]$

nodes: $a = x_0 < x_1 < \dots < x_n = b$

• A cubic spline interpolation S for f is a function that satisfies the conditions:

(a) $S(x) = S_j(x)$: cubic polynomial on $[x_j, x_{j+1}]$

for $j = 0, 1, \dots, n-1$

(b) $S_j(x_j) = f(x_j)$

$S_j(x_{j+1}) = f(x_{j+1})$

for $j = 0, 1, \dots, n-1$

(c) $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$, for $j = 0, 1, \dots, n-2$

(d) $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$, for $j = 0, 1, \ldots, n-2$

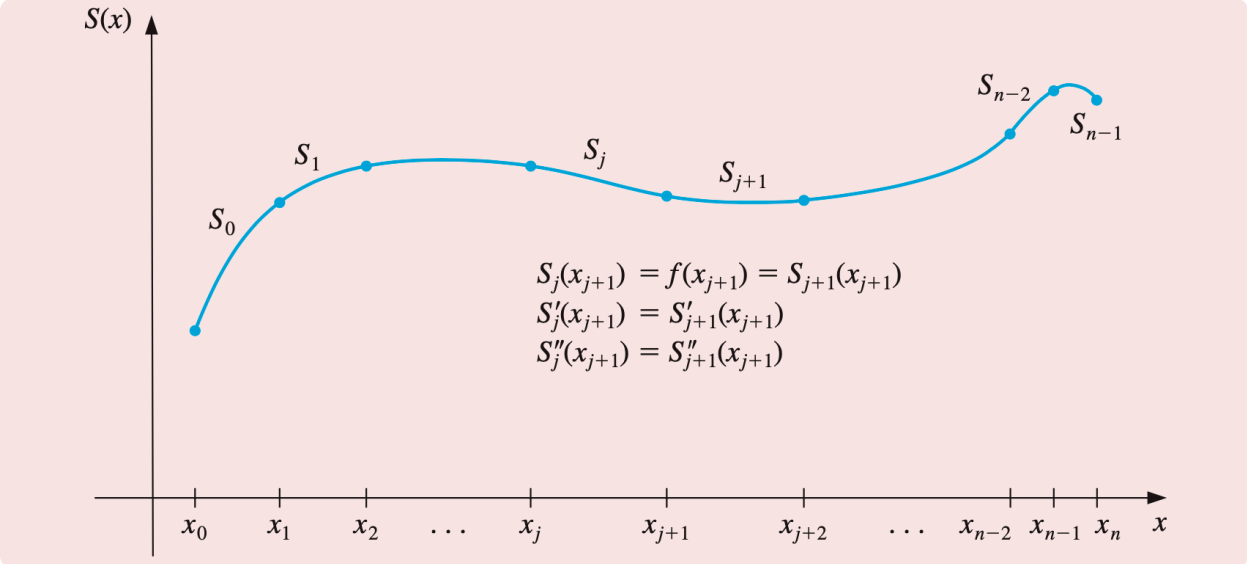
(e) $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$, for $j = 0, 1, \ldots, n-2$

(f) One of the boundary conditions is satisfied

(i) $S''(x_0) = S''(x_n) = 0$ (natural/free boundary)

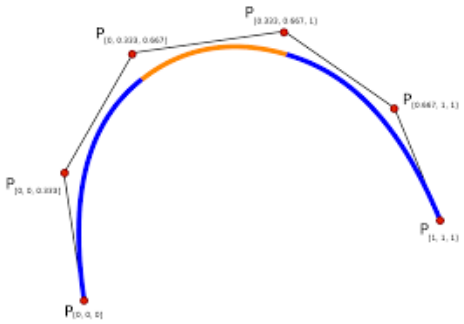
(ii) $\begin{cases} S'(x_0) = f'(x_0) \\ S'(x_n) = f'(x_n) \end{cases}$ (clamped boundary)

{

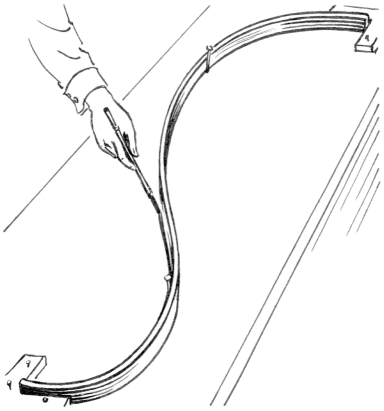


Spline

- "Spline" ~ "Splint"
- It's a small strip of wood that is used to join 2 boards

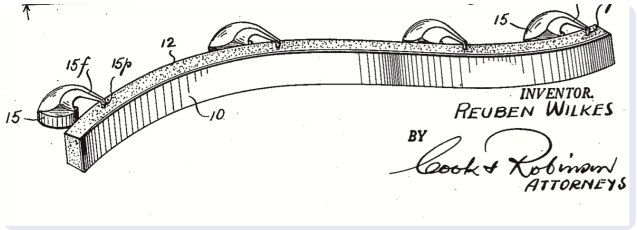


• Later, "spline" is used to refer a long flexible strip that can be used to draw continuous smooth curve by forcing the strip to pass through specified points & tracing along the curve.



Natural Spline

• A natural spline has no conditions imposed for the direction at its endpoints, so the curve takes the shape of a straight line after it passes through the interpolation points nearest its endpoints.



• Clamped boundary conditions ⟶ better approximations but we need more info: $f'(x_0)$ & $f'(x_n)$.

Example

Construct a natural cubic spline through $(1,2)$, $(2,3)$, and $(3,5)$.

Sol

- On $[1,2]$: $S_0(x) = a_0 + b_0(x-1) + c_0(x-1)^2 + d_0(x-1)^3$
- On $[2,3]$: $S_1(x) = a_1 + b_1(x-2) + c_1(x-2)^2 + d_1(x-2)^3$

$$\begin{cases} 2 = f(1) = a_0 \\ 3 = f(2) = a_0 + b_0 + c_0 + d_0 \\ 3 = f(2) = a_1 \\ 5 = f(3) = a_1 + b_1 + c_1 + d_1 \end{cases}$$

- $S_0'(2) = S_1'(2)$, $S_0''(2) = S_1''(2)$

give us

$$\begin{cases} b_0 + 2c_0 + 3d_0 = b_1 \\ 2c_0 + 6d_0 = 2c_1 \end{cases}$$

- Free boundary conditions:

$$S_0''(1) = 0, \qquad S_1''(3) = 0$$

give us

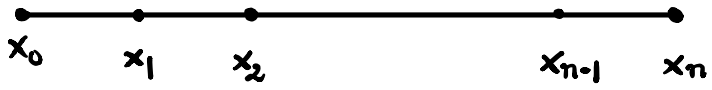
$$\begin{cases} 2c_0 = 0 \\ 2c_1 + 6d_1 = 0 \end{cases}$$

Solving this linear system gives the spline

$$S(x) = \begin{cases} 2 + \frac{3}{4}(x-1) + \frac{1}{4}(x-1)^3, & [1,2] \\ 3 + \frac{3}{2}(x-2) + \frac{3}{4}(x-2)^2 - \frac{1}{4}(x-2)^3, & [2,3] \end{cases}$$

Construction of a Cubic Spline

- Given f and nodes $x_0, x_1, \ldots, x_n$.



n subintervals $[x_j, x_{j+1}]$

- To construct a spline on $[x_0, x_n] \longrightarrow$ we

require $4n$ constants.

- On $[x_j, x_{j+1}]$, we assume that

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

for $j = 0, 1, \ldots, n-1$.

- (b) $\Rightarrow$ $S_j(x_j) = a_j = f(x_j)$.

- (c) $\Rightarrow$ $a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1})$

$$= a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3$$

$$= a_j + b_j h_j + c_j h_j^2 + d_j h_j^3$$

where $h_j := x_{j+1} - x_j$ for $j = 0, 1, \ldots, n-2$.

So $$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \qquad (1)$$

- Define $b_n = S'(x_n)$. Observe that

$$S_j'(x) = b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2$$

so $S_j'(x_j) = b_j$ for $j = 0, 1, \ldots, n-1$.

- (d): $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \Rightarrow$

$$\boxed{b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2} \qquad (2)$$

for $j = 0, 1, \ldots, n-1$

- Define $c_n = S''(x_n)/2$. Observe that

$$S''_j(x) = 2c_j + 6d_j(x - x_j)$$

so $c_j = S''_j(x_j)/2$ for $j = 0, 1, \ldots, n-1$.

- (e): $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) \Rightarrow$

$$2c_{j+1} = 2c_j + 6d_j h_j$$

or equivalently

$$\boxed{c_{j+1} = c_j + 3d_j h_j} \qquad (3)$$

for $j = 0, 1, \ldots, n-1$

$$d_j = \frac{c_{j+1} - c_j}{3h_j}$$

- Substituting this into (1) & (2) gives

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + \frac{h_j^2}{3}(c_{j+1} - c_j)$$

$$\therefore \quad \boxed{a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1})} \qquad (4)$$

and $$\boxed{b_{j+1} = b_j + h_j(c_j + c_{j+1})} \qquad (5)$$

Solve for b_j:

$$\boxed{b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})} \qquad (6)$$

So $$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j)$$

$$b_j = b_{j-1} + h_{j-1}(c_{j-1} + c_j)$$

Thus $$\overbrace{\frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})}^{b_j}$$

$$= \underbrace{\frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j)}_{b_{j-1}} + h_{j-1}(c_{j-1} + c_j)$$

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \quad (7)$$

for $j = 1, 2, \ldots, n-1$

$\longrightarrow$ This linear system involves only $\{c_j\}_{j=0}^{n}$ as unknowns while $\{h_j\}_{j=0}^{n-1}$ and $\{a_j\}_{j=0}^{n}$ are given.

$$\begin{cases} h_j = x_{j+1} - x_j \\ a_j = f(x_j) \end{cases}$$

- Once c_j's are determined, we can find b_j's and d_j's by (6) and (3) respectively.

$\longrightarrow$ We can construct the cubic polynomials $S_j(x)$'s

Question: $\{c_j\}_{j=0}^{n}$ can be found uniquely?

The answer is "yes" if one of the boundary conditions are imposed.

Natural Splines

Theorem

If f is defined at $a = x_0 < x_1 < \dots < x_n = b$ then f has a unique natural spline interpolant S on the nodes $x_0, x_1, \dots, x_n$.

Proof

- Using $S''(x_0) = S''(x_n) = 0$ gives us

$$\begin{cases} 2c_0 + 6d_0(x_0 - x_0) = 0 \\ c_n = S''(x_n)/2 = 0 \end{cases}$$

So $c_0 = c_n = 0$.

- These two equations together with (7) produces the linear system

$$Ax = b$$

$$\downarrow$$

size: $(n+1) \times (n+1)$

$$A = \begin{bmatrix} 1 & 0 & 0 & & & 0 \\ h_0 & 2(h_0+h_1) & h_1 & & & \\ 0 & h_1 & 2(h_1+h_2) & h_2 & & 0 \\ & & & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & \\ 0 & & 0 & 0 & 1 & \end{bmatrix}$$

$$x = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}$$

$A = [a_{ij}]$ is strictly diagonally dominant, i.e.

$$\sum_{j \neq i} |a_{ij}| < |a_{ii}| \implies A \text{ is invertible.}$$

So the system $Ax = b$ has a unique solution.

Pseudo Code for the natural cubic spline S for f defined at $x_0 < x_1 < \dots < x_n$ s.t. $S''(x_0) = S''(x_n) = 0$.

- Input: $n,\ x_0, \dots, x_n$

$$a_0 = f(x_0), \dots, a_n = f(x_n)$$

- Output: a_j, b_j, c_j, d_j for $j = 0, 1, \dots, n-1$.

$$S(x) = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

for $x_j \leq x \leq x_{j+1}$

- For $i \leftarrow 0$ to $n-1$ do

$$h_i = x_{i+1} - x_i$$

- For $i \leftarrow 1$ to $n-1$ do

$$\alpha_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$$

- Solve a tridiagonal linear system

Set $l_0 = 1$, $\mu_0 = 0$, $z_0 = 0$

- For $i \leftarrow 1$ to $n-1$ do

$$l_i = 2(x_i - x_{i-1}) - h_{i-1}\mu_{i-1}$$

$$\mu_i = h_i / l_i$$

$$z_i = (\alpha_i - h_{i-1} z_{i-1}) / l_i$$

- Set $l_n = 1$, $z_n = 0$, $c_n = 0$

- For $j \leftarrow n-1$ to 0 do

$$c_j = z_j - \mu_j c_{j+1}$$

$$b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$$

$$d_j = (c_{j+1} - c_j)/(3h_j)$$

- Return a_j, b_j, c_j, d_j

Example:

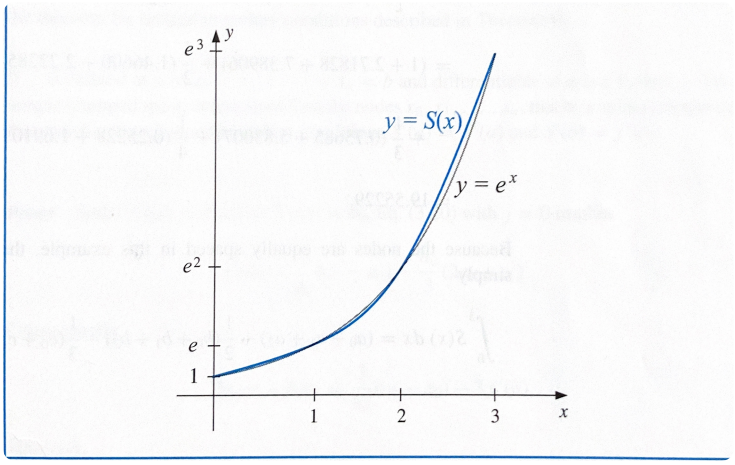
Use the data points $(0,1)$, $(1,e)$, $(2,e^2)$, and $(3,e^3)$ to form a natural spline $S(x)$ that approximates

$f(x) = e^x$

```python
import numpy as np
from math import *

def natural_cubic_spline(mydata):
    n = len(mydata)
    a = np.zeros(n,float)
    b = np.zeros(n,float)
    c = np.zeros(n,float)
    d = np.zeros(n,float)
    h = np.zeros(n,float)
    l = np.zeros(n,float)
    mu = np.zeros(n,float)
    z = np.zeros(n,float)
    alpha = np.zeros(n,float)
    for i in range(n):
        a[i] = mydata[i][1]
    for i in range(n-1):
        h[i] = mydata[i+1][0] - mydata[i][0]
    for i in range(1,n-1):
        alpha[i] = (3/h[i])*(a[i+1]-a[i])-(3/h[i-1])*(a[i]-a[i-1])
    l[0] = 1
    for i in range(1,n-1):
        l[i] = 2*(mydata[i+1][0]-mydata[i-1][0])-h[i-1]*mu[i-1]
        mu[i] = h[i]/l[i]
        z[i] = (alpha[i]-h[i-1]*z[i-1])/l[i]
    l[n-1] = 1
    j = n-2
    while True:
        c[j] = z[j] - mu[j]*c[j+1]
        b[j] = (a[j+1] - a[j])/h[j] - h[j]*(c[j+1]+2*c[j])/3
        d[j] = (c[j+1]-c[j])/(3*h[j])
        j -=1
        if j == -1:
            break
    return a, b, c, d

mydata1 = np.array( [ [1,2], [2,3], [3,5] ])
mydata2 = np.array( [ [0,1], [1,exp(1)], [2,exp(2)], [3,exp(3)] ] )

natural_cubic_spline(mydata2)
```

Running this code gives us

$$S(x) = \begin{cases} 1 + 1.47x + 0.25x^3 & [0,1] \\ 2.72 + 2.22(x-1) + 0.76(x-1)^2 + 1.69(x-1)^3, & [1,2] \\ 7.39 + 8.81(x-2) + 5.83(x-2)^2 - 1.94(x-2)^3, & [2,3] \end{cases}$$



Clamped Splines

Theorem

If f is defined at $a = x_0 < x_1 < \dots < x_n = b$ then f has a unique clamped spline interpolant S on the nodes $x_0, \dots, x_n$.

Proof

Using $S'(x_0) = f'(x_0) = f'(a) = b_0$ gives us

$$f'(a) = \frac{1}{h_0}(a_1 - a_0) - \frac{h_0}{3}(2c_0 + c_1)$$

thanks to (6).

So

$$2h_0 c_0 + h_0 c_1 = \frac{3}{h_0}(a_1 - a_0) - 3f'(a). \quad (8)$$

Similarly,

$$f'(b) = b_n = b_{n-1} + h_{n-1}(c_{n-1} + c_n).$$

Combining this with (6) for $j = n-1$ we get

$$f'(b) = \frac{a_n - a_{n-1}}{h_{n-1}} - \frac{h_{n-1}}{3}(2c_{n-1} + c_n) + h_{n-1}(c_{n-1} + c_n)$$

$$= \frac{a_n - a_{n-1}}{h_{n-1}} + \frac{h_{n-1}}{3}(c_{n-1} + 2c_n)$$

which implies that

$$h_{n-1} c_{n-1} + 2h_{n-1} c_n = 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1})$$

Combining this with (7) and (8) gives us the linear system $Ax = b$, where

$$A = \begin{bmatrix} 2h_0 & h_0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & & \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \\ & & \ddots & \ddots & h_{n-2} \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \\ 0 & & & h_{n-1} & 2h_{n-1} \end{bmatrix}$$

$$b = \begin{bmatrix} \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{bmatrix}, \quad x = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

A is also strictly diagonally dominant, so the linear system has a unique solution. ☺

Pseudo Code for the clamped cubic spline S for f defined at $x_0 < x_1 < \dots < x_n$ s.t. $f'(x_0) = S'(x_0)$ & $f'(x_n) = S'(x_n)$.

- Input: $n, x_0, \dots, x_n$

  $a = f(x_0), \dots, b = f(x_n)$

  $FPO = f'(x_0), \quad FPN = f'(x_n)$

- Output: a_j, b_j, c_j, d_j for $j = 0, 1, \dots, n-1$

$S(x) = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$

- For $i \leftarrow 0$ to $n-1$ do

  $h_i = x_{i+1} - x_i$

- Set $\alpha_0 = 3(a_1 - a_0)/h_0 - 3FPO$

  $\alpha_n = 3FPN - 3(a_n - a_{n-1})/h_{n-1}$

- For $i \leftarrow 1$ to $n-1$ do

  $$\alpha_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$$

- Set $l_0 = 2h_0$,

$\mu_0 = 0.5,$

$z_0 = \alpha_0 / l_0$

- For $i \leftarrow 1$ to $n-1$ do

  $l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$

  $\mu_i = h_i / l_i$

  $z_i = (\alpha_i - h_{i-1} z_{i-1}) / l_i$

- Set $l_{n-1} = h_{n-1}(2 - \mu_{n-1})$

  $z_n = (\alpha_n - h_{n-1} z_{n-1}) / l_n$

  $c_n = z_n$

- For $j \leftarrow n-1$ to 0 do

  set $c_j = z_j - \mu_j c_{j+1}$

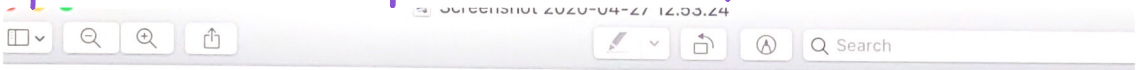
  $b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$

  $d_j = (c_{j+1} - c_j)/(3h_j)$

- Return a_j, b_j, c_j, d_j

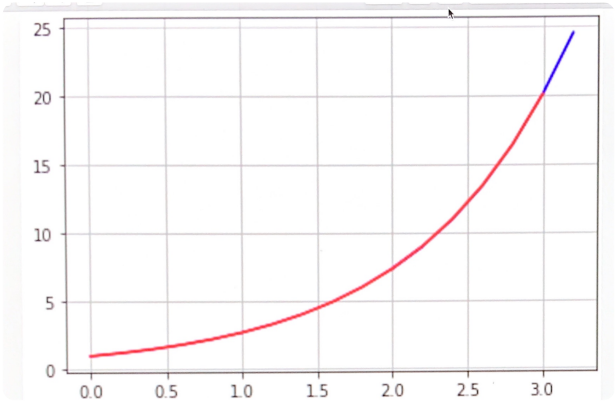
Example:

Use the data points $(0,1)$, $(1,e)$, $(2,e^2)$, and $(3,e^3)$ to form a clamped spline $S(x)$ that approximates $f(x) = e^x$ with $f'(0) = 1$ and $f'(3) = e^3$.




```python
import numpy as np
from math import *

def clamped_cubic_spline(mydata):
    n = len(mydata)
    a = np.zeros(n,float)
    b = np.zeros(n,float)
    c = np.zeros(n,float)
    d = np.zeros(n,float)
    fpo = mydata[0][2]
    fpn = mydata[n-1][2]
    h = np.zeros(n,float)
    l = np.zeros(n,float)
    mu = np.zeros(n,float)
    z = np.zeros(n,float)
    alpha = np.zeros(n,float)
    for i in range(n):
        a[i] = mydata[i][1]
    for i in range(n-1):
        h[i] = mydata[i+1][0] - mydata[i][0]

    alpha[0] = 3*(a[1]-a[0])/h[0] - 3*fpo
    alpha[n-1] = 3*fpn - 3*(a[n-1]-a[n-2])/h[n-2]
    for i in range(1,n-1):
        alpha[i] = (3/h[i])*(a[i+1]-a[i])-(3/h[i-1])*(a[i]-a[i-1])
    l[0] = 2*h[0]
    mu[0] = 0.5
    z[0] = alpha[0]/l[0]
    for i in range(1,n-1):
        l[i] = 2*(mydata[i+1][0]-mydata[i-1][0])-h[i-1]*mu[i-1]
        mu[i] = h[i]/l[i]
        z[i] = (alpha[i]-h[i-1]*z[i-1])/l[i]

    l[n-1] = h[n-2]*(2-mu[n-2])
    z[n-1] = (alpha[n-1]-h[n-2]*z[n-2])/l[n-1]
    c[n-1] = z[n-1]
    j = n-2
    while True:
        c[j] = z[j] - mu[j]*c[j+1]
        b[j] = (a[j+1] - a[j])/h[j] - h[j]*(c[j+1]+2*c[j])/3
        d[j] = (c[j+1]-c[j])/(3*h[j])
        j -=1
        if j == -1:
            break
    return a, b, c, d

mydata = np.array( [ [0,1,1], [1,exp(1),exp(1)], [2,exp(2),exp(2)], [3,exp(3),exp(3)] ] )
clamped_cubic_spline(mydata)
```

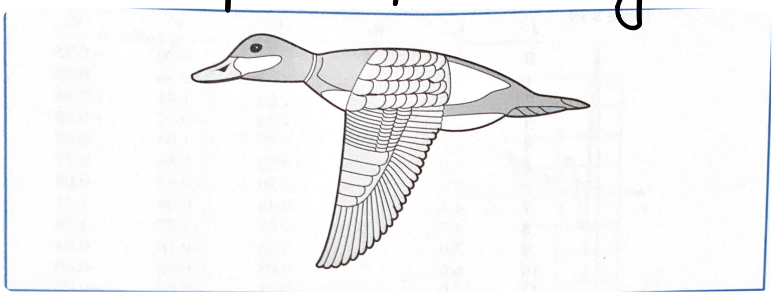
$$S(x)=\begin{cases} 1 + x + 0.45x^2 + 0.27x^3, & [0,1) \\ 2.72 + 2.71(x-1) + 1.27(x-1)^2 + 0.69(x-1)^3, & [1,2) \\ 7.39 + 7.33(x-2) + 3.35(x-2)^2 + 2.02(x-2)^3, & [2,3) \end{cases}$$



a better approximation

Illustration:

Suppose we have a picture of a ruddy duck in flight.



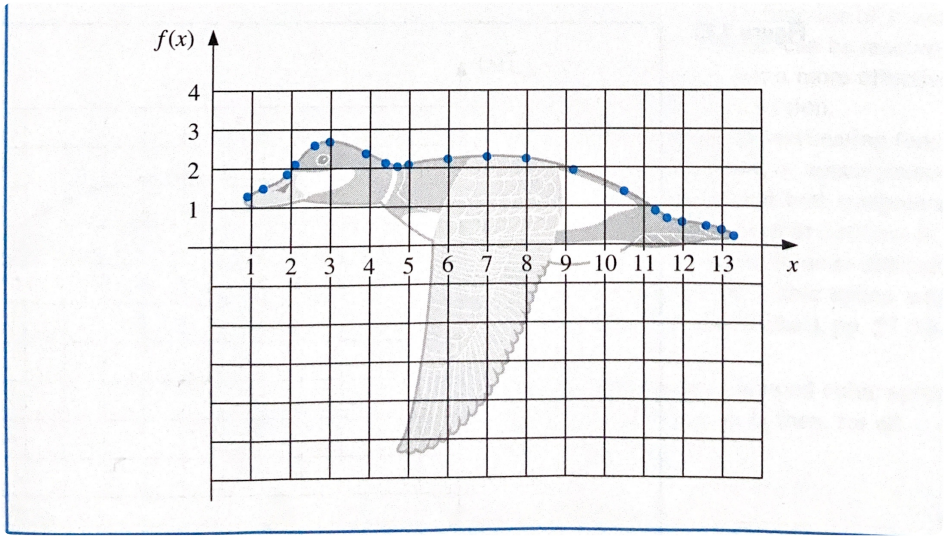
To approximate the top profile of the duck, we choose

points along the curve through what we want the approximating curve to pass.

To proceed, we list 21 data points as follows

```
mydata3 = np.array([ [0.9, 1.3], [1.3, 1.5],
                     [1.9, 1.85], [2.1, 2.1],
                     [2.6, 2.6], [3.0, 2.7],
                     [3.9, 2.4], [4.4, 2.15],
                     [4.7, 2.05], [5.0, 2.1],
                     [6.0, 2.25], [7.0,2.3],
                     [8.0, 2.25], [9.2, 1.95],
                     [10.5, 1.4], [11.3, 0.9],
                     [11.6, 0.7], [12.0, 0.6],
                     [12.6, 0.5], [13.0, 0.4],
                     [13.3, 0.25]])
```

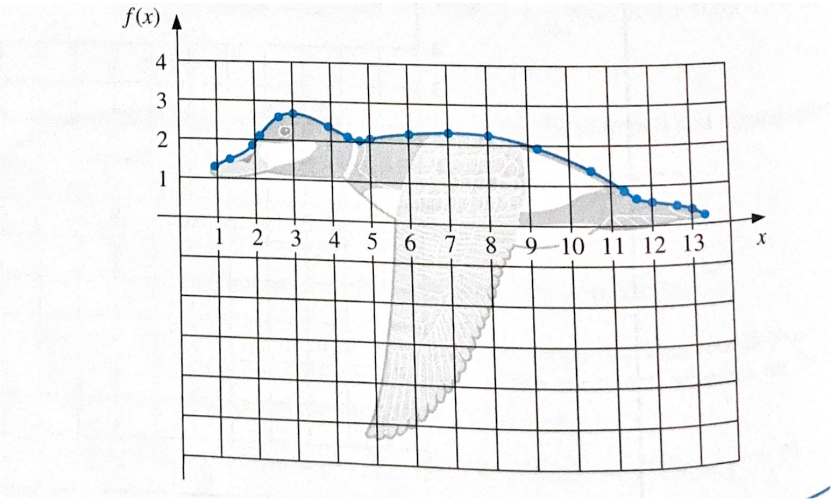
which can be plotted as follows



Using the written code in Python to generate the natural cubic spline for these data points produces

the following coefficients

j	x_j	a_j	b_j	c_j	d_j
0	0.9	1.3	0.54	0.00	−0.25
1	1.3	1.5	0.42	−0.30	0.95
2	1.9	1.85	1.09	1.41	−2.96
3	2.1	2.1	1.29	−0.37	−0.45
4	2.6	2.6	0.59	−1.04	0.45
5	3.0	2.7	−0.02	−0.50	0.17
6	3.9	2.4	−0.50	−0.03	0.08
7	4.4	2.15	−0.48	0.08	1.31
8	4.7	2.05	−0.07	1.27	−1.58
9	5.0	2.1	0.26	−0.16	0.04
10	6.0	2.25	0.08	−0.03	0.00
11	7.0	2.3	0.01	−0.04	−0.02
12	8.0	2.25	−0.14	−0.11	0.02
13	9.2	1.95	−0.34	−0.05	−0.01
14	10.5	1.4	−0.53	−0.10	−0.02
15	11.3	0.9	−0.73	−0.15	1.21
16	11.6	0.7	−0.49	0.94	−0.84
17	12.0	0.6	−0.14	−0.06	0.04
18	12.6	0.5	−0.18	0.00	−0.45
19	13.0	0.4	−0.39	−0.54	0.60
20	13.3	0.25			



- When the nodes are equally spaced near both endpoints, approximations can be obtained by appropriate formulas that we will study later.
- When the nodes are unequally spaced, the problem is very challenging.

The error-bound formula for the cubic spline with clamped boundary conditions.

Theorem:

$f \in C^4([a,b])$, $\max_{[a,b]} |f^{(4)}(x)| = M$.

If S is the unique clamped cubic spline interpolant to f w.r.t. the nodes $a = x_0 < x_1 < \dots < x_n = b$, then, for all $x \in [a,b]$,

$$|f(x) - S(x)| \leq \frac{5M}{384} \max_{0 \leq j \leq n-1} (x_{j+1} - x_j)^4$$