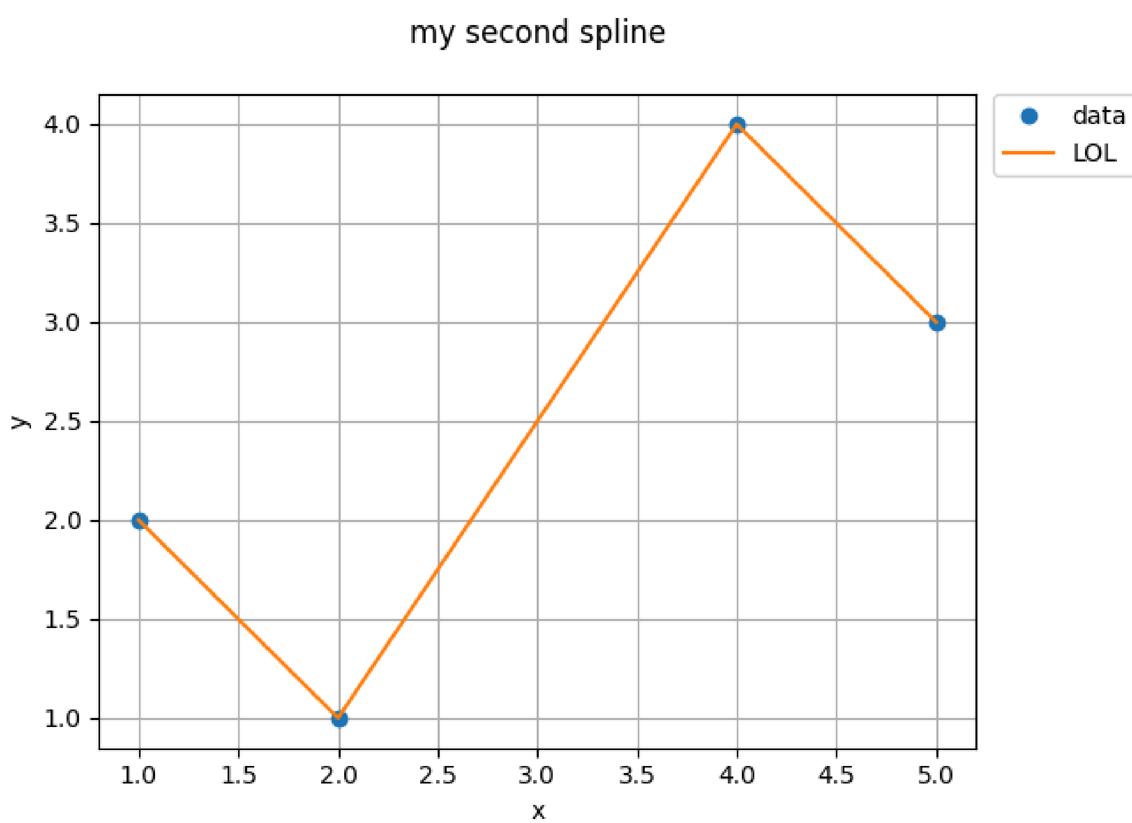
03.04 cubic spines

so theres interpolating polynomials for (sort-of) random x_i, evenly spaced x_i and chebyshev x_i. what about other ways to connect the dots? what if the word "continuous" gets taken away?

code, visual

```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp

#https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html

def main():⋯

if __name__ == "__main__":⋯
```



usw

no ones gonna pay money for that but lets write it out seriously:

$$
\begin{aligned}
S_1(x) &= 2-(x-1) \quad \text{on } [1,2] \\
S_2(x) &= 1+\frac{3}{2}(x-2) \quad \text{on } [2,4] \\
S_3(x) &= 4-(x-4) \quad \text{on } [4,5].
\end{aligned}
$$

and if it smoothed out in a cubic-spline kind of way,

$$
\begin{aligned}
S_1(x) &= 2-\frac{13}{8}(x-1)+0\cdot(x-1)^2+\frac{5}{8}(x-1)^3 \quad \text{on } [1,2] \\
S_2(x) &= 1+\frac{1}{4}(x-2)+\frac{15}{8}(x-2)^2-\frac{5}{8}(x-2)^3 \quad \text{on } [2,4] \\
S_3(x) &= 4+\frac{1}{4}(x-4)-\frac{15}{8}(x-4)^2+\frac{5}{8}(x-4)^3 \quad \text{on } [4,5].
\end{aligned}
$$

note the smooth transition from one S_i to the next at the middle base points, or "knots". this is achieved by having neighboring pieces S_i, S_{i+1} have the same zeroth, first and second derivatives at those points.

given n points $(x_1, y_1), \ldots, (x_n, y_n)$, there is only one linear spline but there are infinitely many cubic splines. ie, more conditions are needed to decide the spline.

1 properties of splines

assume the n given data points $(x_1, y_1), \ldots, (x_n, y_n)$ are of distinct x_i and in increasing order. a **cubic spline** $S(x)$ through data points $(x_1, y_1), \ldots, (x_n, y_n)$ is a set of cubic polynomials.

$$
\begin{aligned}
S_1(x) &= y_1+b_1(x-x_1)+c_1(x-x_1)^2+d_1(x-x_1)^3 \quad \text{on } [x_1, x_2] \\
S_2(x) &= y_2+b_2(x-x_2)+c_2(x-x_2)^2+d_2(x-x_2)^3 \quad \text{on } [x_2, x_3] \\
&\vdots \\
S_{n-1}(x) &= y_{n-1}+b_{n-1}(x-x_{n-1})+c_{n-1}(x-x_{n-1})^2+d_{n-1}(x-x_{n-1})^3 \quad \text{on } [x_{n-1}, x_n]
\end{aligned}
$$

with the following properties:

property 01 $S_i(x_i) = y_i, S_i(x_{i+1}) = y_{i+1}$ for $i = 1, \ldots, n-1$.

property 02 $S'_{i-1}(x_i) = S_i(x_i)'$ for $i = 2, \ldots, n-1$.

property 03 $S''_{i-1}(x_i) = S_i(x_i)''$ for $i = 2, \ldots, n-1$.

property 01 guarantees that spline $S(x)$ interpolates the data points; property 02 forces the slopes neighboring segments to agree where they meet; property 03 does the same wrt curvature.

property 04a natural spline $S_1''(x_1) = 0, S_{n-1}''(x_n) = 0.$

n points will have $n-1$ segments, each represented by a cubic and each cubic will have a coefficient for each of its term. term $a_0 x^0 = y_i$, leaving $b_i \equiv a_2, c_i \equiv a_i, d_i \equiv a_3$ per equation i.

property 1 at $x_{i+1} \to n-1$ equations:

$$\begin{aligned} y_2 &= S_1(x_2) = y_1 + b_1(x_2 - x_1) + c_1(x_2 - x_1)^2 + d_1(x_2 - x_1)^3 \\ &\vdots \\ y_n &= S_{n-1}(x_n) = y_{n-1} + b_{n-1}(x_n - x_{n-1}) + c_{n-1}(x_n - x_{n-1})^2 + d_{n-1}(x_n - x_{n-1})^3. \end{aligned}$$

property 02 $\to n-2$ equations :

$$\begin{aligned} 0 &= S_1'(x_2) - S_2'(x_2) = b_1 + 2c_1(x_2 - x_1) + 3d_1(x_2 - x_1)^2 - b_2 \\ &\vdots \\ 0 &= S_{n-2}'(x_{n-1}) - S_{n-1}'(x_{n-1}) = b_{n-2} + 2c_{n-2}(x_{n-1} - x_{n-2}) + 3d_{n-2}(x_{n-1} - x_{n-2})^2 - b_{n-1}. \end{aligned}$$

property 03 $\to n-2$ equations :

$$\begin{aligned} 0 &= S_1''(x_2) - S_2''(x_2) = 2c_1 + 6d_1(x_2 - x_1) - 2c_2 \\ &\vdots \\ 0 &= S_{n-2}''(x_{n-1}) - S_{n-1}''(x_{n-1}) = 2c_{n-2} + 6d_{n-2}(x_{n-1} - x_{n-2}) - 2c_{n-1}. \end{aligned}$$

instead of solving $3 \times (n-1)$ equations all at once, break it down. find c_i within a subset of those equations, then solve out from there.

consider $c_n = S_{n-1}''(x_n) \cdot \frac{1}{2}$. let $\delta_i = x_{i+1} - x_i, \Delta_i = y_{i-1} - y_i$. then property 03 equations resolve to

$$d_i = \frac{c_{i+1} - c_i}{3\delta_i} \quad i = 1, \dots, n-1$$

and property 01 equations reduce to

$$\begin{aligned} b_i &= \frac{\Delta_i}{\delta_i} - c_i\delta_i - d_i\delta_i^2 \\ &= \frac{\Delta_i}{\delta_i} - c_i\delta_i - \frac{\delta_i}{3}(c_{i-1} - c_i) \\ &= \frac{\Delta_i}{\delta_i} - \frac{\delta_i}{3}(2c_i - c_{i+1}) \quad i = 1, \dots, n-1. \end{aligned}$$

ie, both b, d are both in terms of c. so

$$\begin{aligned} \delta_1 c_1 + 2(\delta_1 + \delta_2)c_2 + \delta_2 c_3 &= 3\left(\frac{\Delta_2}{\delta_2} - \frac{\Delta_1}{\delta_1}\right) \\ &\vdots \\ \delta_{n-2}c_{n-2} + 2(\delta_{n-2} + \delta_{n-1})c_{n-1} + \delta_{n-1}c_n &= 3\left(\frac{\Delta_{n-1}}{\delta_{n-1}} - \frac{\Delta_{n-2}}{\delta_{n-2}}\right). \end{aligned}$$

property 04a, natural splines $\rightarrow$ 2 equations:

$$\begin{aligned} S_1''(x_1) = 0 \rightarrow 2c_1 &= 0 \\ S_{n-1}''(x_n) = 0 \rightarrow 2c_n &= 0. \end{aligned}$$

n equations, n unknowns in c_i,

$$\begin{bmatrix} 1 & 0 & 0 & & & \\ \delta_1 & 2\delta_1 + 2\delta_2 & \delta_2 & \ddots & & \\ 0 & \delta_2 & 2\delta_2 + 2\delta_3 & \delta_3 & \ddots & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & & \delta_{n-2} & 2\delta_{n-2} + 2\delta_{n-1} & \delta_{n-1} \\ & & & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ \\ \vdots \\ \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 3\left(\frac{\Delta_2}{\delta_2} - \frac{\Delta_1}{\delta_1}\right) \\ \vdots \\ 3\left(\frac{\Delta_{n-1}}{\delta_{n-1}} - \frac{\Delta_{n-2}}{\delta_{n-2}}\right) \\ 0 \end{bmatrix}.$$

then solve for b_i, d_i directly from c_i. which proves the following theorem.

theorem 07 for a set of data points $(x_1, y_1), \dots, (x_n, y_n)$ with distinct x_i, there is a unique natural cubic spline fitting the points.

natural cubic spline

```
# given x = [x(1),...,x(n)] where x(1)<...<x(n),y=[y(1),...,y(n)]
```

```
for i = 1 : n-1
  a(i) = y(i)
  δ(i) = x(i+1) - x(i)
  Δ(i) = y(i+1) - y(i)
end

solve for c(1),...,c(n)

for i = 1 : n-1
  d(i) = (c(i+1)-c(i))/(3*δ(i))
  b(i) = Δ(i)/δ(i) - (2c(i)+c(i+1))*δ(i)/3
end
```

the natural cubic spline is

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \text{ on } [x_i, x_{i+1}] \text{ for } i = 1, \ldots, n-1.$$

example 14

find natural cubic spline through $(0, 3), (1, -2), (2 - 1)$.

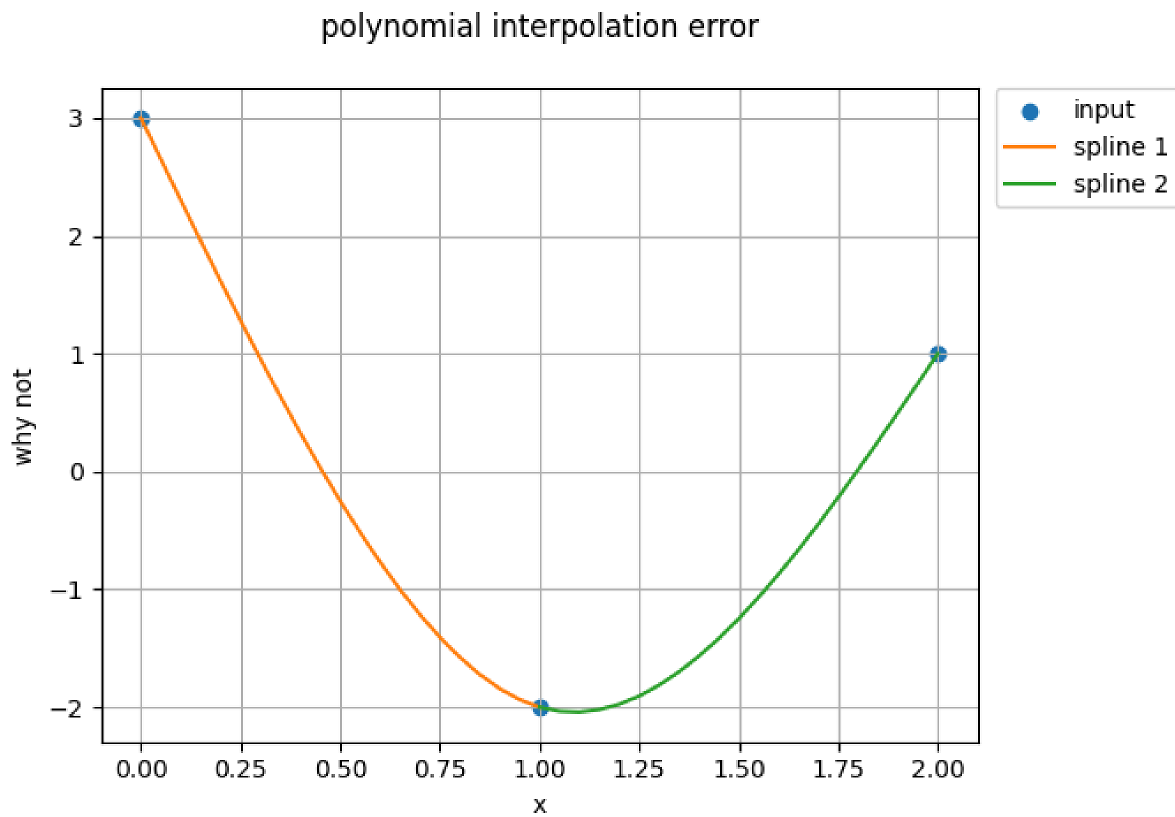
for c_i where $n = 3$,

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 4 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 24 \\ 0 \end{bmatrix}$$

code

```
# bc im even lazier

import matplotlib.pyplot as plt
import numpy as np
import numpy.polynomial as npp

def main():
  #input = [[0,3],[1,-2],[2,1]]
  xs = [0,1,2]
  ys = [3,-2,1]
```

```
spline 1 :

3.0 - 7.0·x + 0.0·x² + 2.0·x³

spline 2 :

          3            2
-2 (x-1) + 6 (x-1)  - 1 (x-1) - 2
```



2 endpoint conditions

property 04b curvature-adjusted cubic spline $S_1''(x_1), S_{n-1}''(x_n)$ set to arbitrary non-zero values.

property 04c clamped cubic spline $S_1'(x_1), S_{n-1}'(x_n)$ set to arbitrary non-zero values.

property 04d parabolically terminated cubic spline. first and last points are forced to be at most degree two by specifying $d_1 = 0 = d_{n-1}$. equivalently, we can require $c_1 = c_2, c_{n-1} = c_n$, which reduces the matrix equation to a strictly diagonally dominant $n-2 \times n-2$ matrix equation in c_2, c_{n-1}.

property 04e not-a-knot cubic spline $S_1'''(x_2) = S_2'''(x_2), S_{n-2}'''(x_{n-1}) = S_{n-1}'''(x_{n-1})$ set to arbitrary non-zero values. ie, the third derivatives here cause $S_1 = S_2, S_{n-2} = S_{n-1}$. eg, this means x_2 doesnt need to be a base point bc $S_2 = S_1$.

theorem 08

assume $n \geq 2$. then for set of data points $(x_1, y_1), \ldots, (x_n, y_n)$ and for any one of end conditions given by properties 4a-4c, there is a unique cubic spline satisfying the end conditions and fitting the points. the same is true assuming $n \geq 3$ for property 4d and $n \geq 4$ for 4e.

code

```
# bc im laziest = hack of previous to include parabolic end conditions

import numpy as np
import numpy.polynomial as npp

def parabolic_cubic_spline(xs,ys,ni):
  """
  xs : x-data
  ys : y-data
  ni : number of model evaluation points for each spline
```

```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp

#https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html

def main():⋯

if __name__ == "__main__":
  main()
```