Computer Methods (MAE 3403)

Interpolation

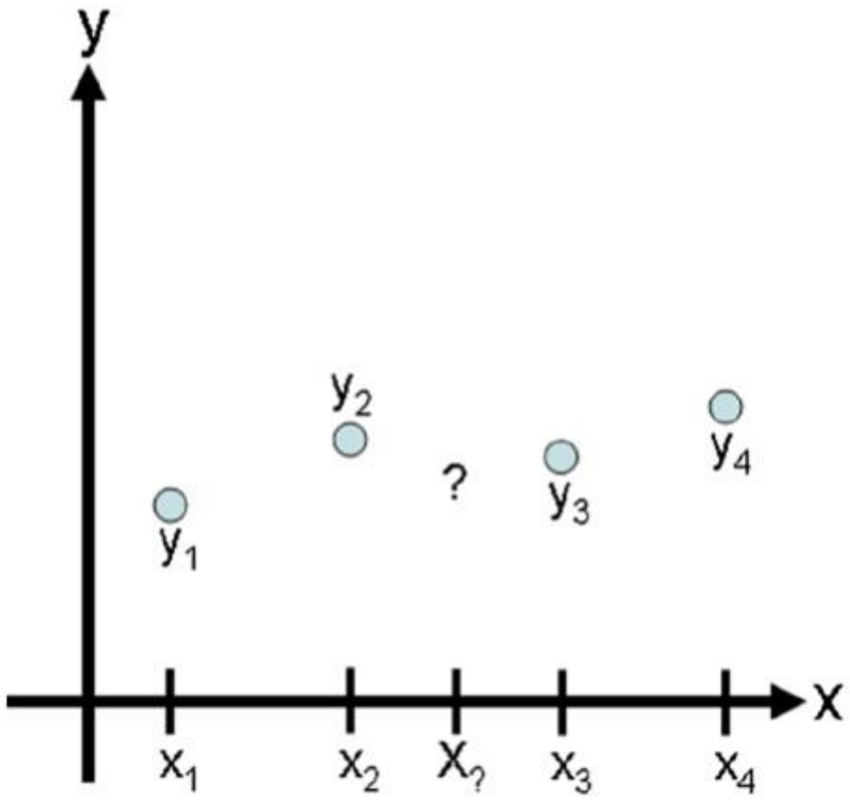
Motivation

- Least squares regression assumes errors in the data and finds the best fit in the squared error sense
- Good data: estimate a function that goes through the data points?
- Interpolation does that.

Interpolation

- Given a data set of x_i and y_i, $i=1,\ldots,n$, find an estimation function $\hat{y}(x)$ such that $\hat{y}(x_i) = y_i$.

- If there is a new x^*, we can predict $y^* = \hat{y}(x^*)$
  - x^* is in the range of x_i's



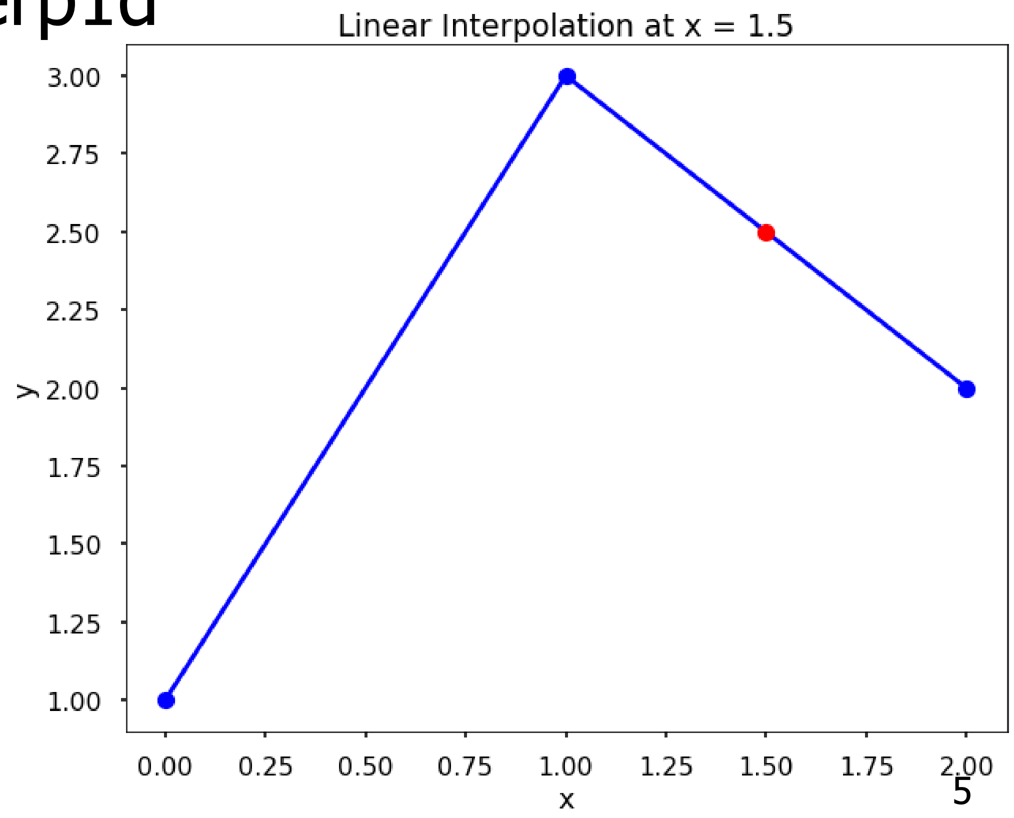
Linear interpolation

- The estimated point is assumed to lie on the line joining the nearest points to the left and right.
- Say $x_i<x<x_{i+1}$, then the interpolated value at x is $y_i+(y_{i+1} - y_i)(x-x_i)/(x_{i+1}-x_i)$
- Example: find the linear interpolation at x=1.5 based on data x=[0,1,2] and y=[1,3,2]

Python example

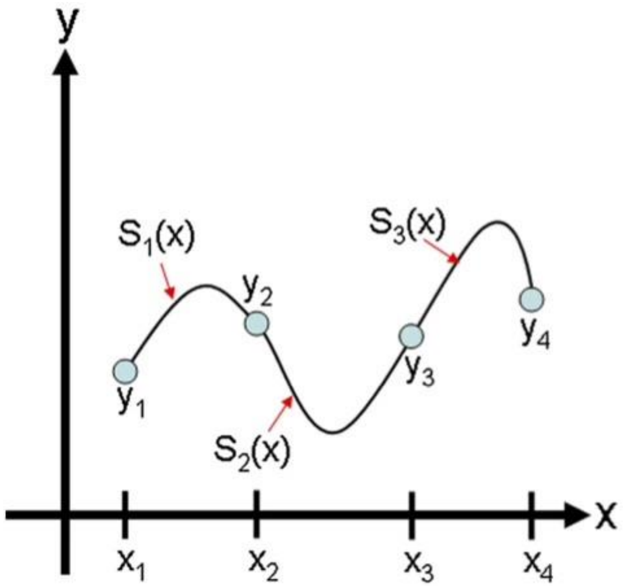
- scipy.interpolate has various interpolation functions

```
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
x = [0, 1, 2]
y = [1, 3, 2]
f = interp1d(x, y)
y_hat = f(1.5)
print(y_hat)
```



Cubic Spline interpolation

- The interpolating function is a set of cubic functions, i.e., the points (x_i, y_i) and (x_{i+1}, y_{i+1}) are joined via a cubic polynomial $S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$ for the region $[x_i, x_{i+1}]$.
- *n data points, how many cubic functions? how many total unknown parameters? how many independent equations needed?*



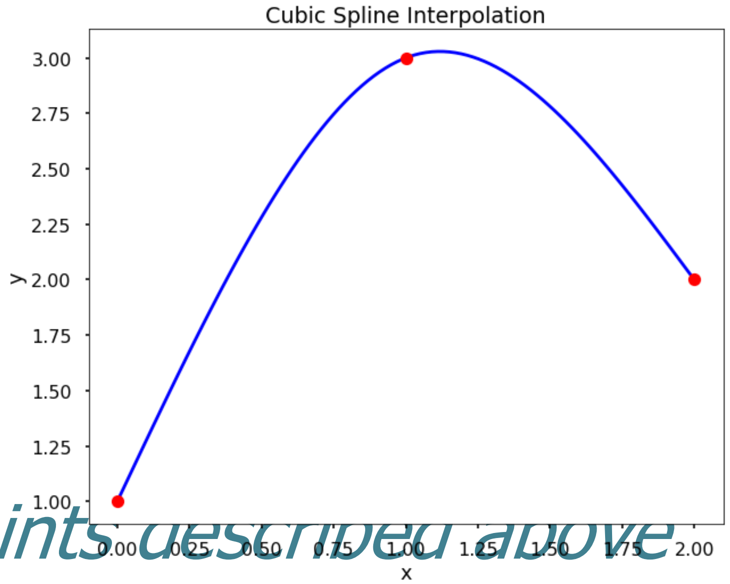
Answers

- *Given n data points, how many cubic functions?*
  - *n-1*
- *How many total unknown parameters?*
  - *4*(n-1)*
- *how many independent equations needed?*
  - *4*(n-1)*

Construction

- First, cubic functions must intersect the data points on the left and right
  - $S_i(x_i)=y_i$, $S_i(x_{i+1}) = y_{i+1}$, $i = 1,\ldots,n-1$
- Second, each cubic function joins smoothly with its neighbors: continuous 1st and 2nd derivatives
  - $S'_i(x_{i+1})=S'_{i+1}(x_{i+1})$, $S''_i(x_{i+1})=S''_{i+1}(x_{i+1})$, $i = 1,\ldots,n-2$
- Two more constraints: arbitrary, e.g., 2nd derivatives are zero at the endpoints: $S''_1(x_1) = S''_{n-1}(x_n) = 0$.

Python implementation: *CubicSpline*




```
from scipy.interpolate import CubicSpline
import numpy as np
import matplotlib.pyplot as plt
x = [0, 1, 2]
y = [1, 3, 2]
# use bc_type = 'natural' adds the constraints described above
f = CubicSpline(x, y, bc_type='natural’)
x_new = np.linspace(0, 2, 100)
y_new = f(x_new)
```

HW: code CubicSpline yourself

- The key is to create the A and b matrices to solve for the unknown coefficients (a_i, b_i, c_i, d_i, $i=1,\ldots n-1$) in the cubic polynomials ($S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$, $i=1,\ldots,n-1$)
- First set of constraints:
  - $S_i(x_i)=y_i$, $S_i(x_{i+1}) = y_{i+1}$, $i = 1,\ldots,n-1$
  - $a_1x_1^3 + b_1x_1^2 + c_1x_1 + d_1 = y_1$, …,
    $a_{n-1}x_{n-1}^3 + b_{n-1}x_{n-1}^2 + c_{n-1}x_{n-1} + d_{n-1} = y_{n-1}$
  - $a_1x_2^3 + b_1x_2^2 + c_1x_2 + d_1 = y_2$, …,
    $a_{n-1}x_n^3 + b_{n-1}x_n^2 + c_{n-1}x_n + d_{n-1} = y_n$

2^{nd} and 3^{rd} sets of constraints

- $S'_i(x_{i+1})=S'_{i+1}(x_{i+1})$, $S''_i(x_{i+1})=S''_{i+1}(x_{i+1})$, $i = 1,\dots,n-2$
  - $3a_1x_2^2 + 2b_1x_2 + c_1 - 3a_2x_2^2 - 2b_2x_2 - c_2=0, \dots$
    $3a_{n-2}x_{n-1}^2 + 2b_{n-2}x_{n-1} + c_{n-2} - 3a_{n-1}x_{n-1}^2 - 2b_{n-1}x_{n-1} - c_{n-1}$
  - $6a_1x_2 + 2b_1 - 6a_2x_2 - 2b_2=0$
    $6a_{n-2}x_{n-1} + 2b_{n-2} - 6a_{n-1}x_{n-1} - 2b_{n-1}=0$
- $S''_1(x_1) = S''_{n-1}(x_n) = 0$
  - $6a_1x_1 + 2b_1=0$
  - $6a_{n-1}x_n + 2b_{n-1}=0$

Putting everything together as Ax = b

$$x = [\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1, \mathrm{d}_1, \mathrm{a}_2, \mathrm{b}_2, \mathrm{c}_2, \mathrm{d}_2, \cdots, \mathrm{a}_{\mathrm{n}-1}, \mathrm{b}_{\mathrm{n}-1}, \mathrm{c}_{\mathrm{n}-1}, \mathrm{d}_{n-1}]^{\mathrm{T}}$$

Lagrange Polynomial Interpolation

- Cubic spline: joins multiple cubic polynomials

- Lagrange polynomial *L(x)*: finds a single polynomial that goes through all points.

$$L(x) = \sum_{i=1}^{n} y_i P_i(x)$$

$$P_i(x) = \prod_{j=1, j \neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

- Can you verify $L(x_i) = y_i$?

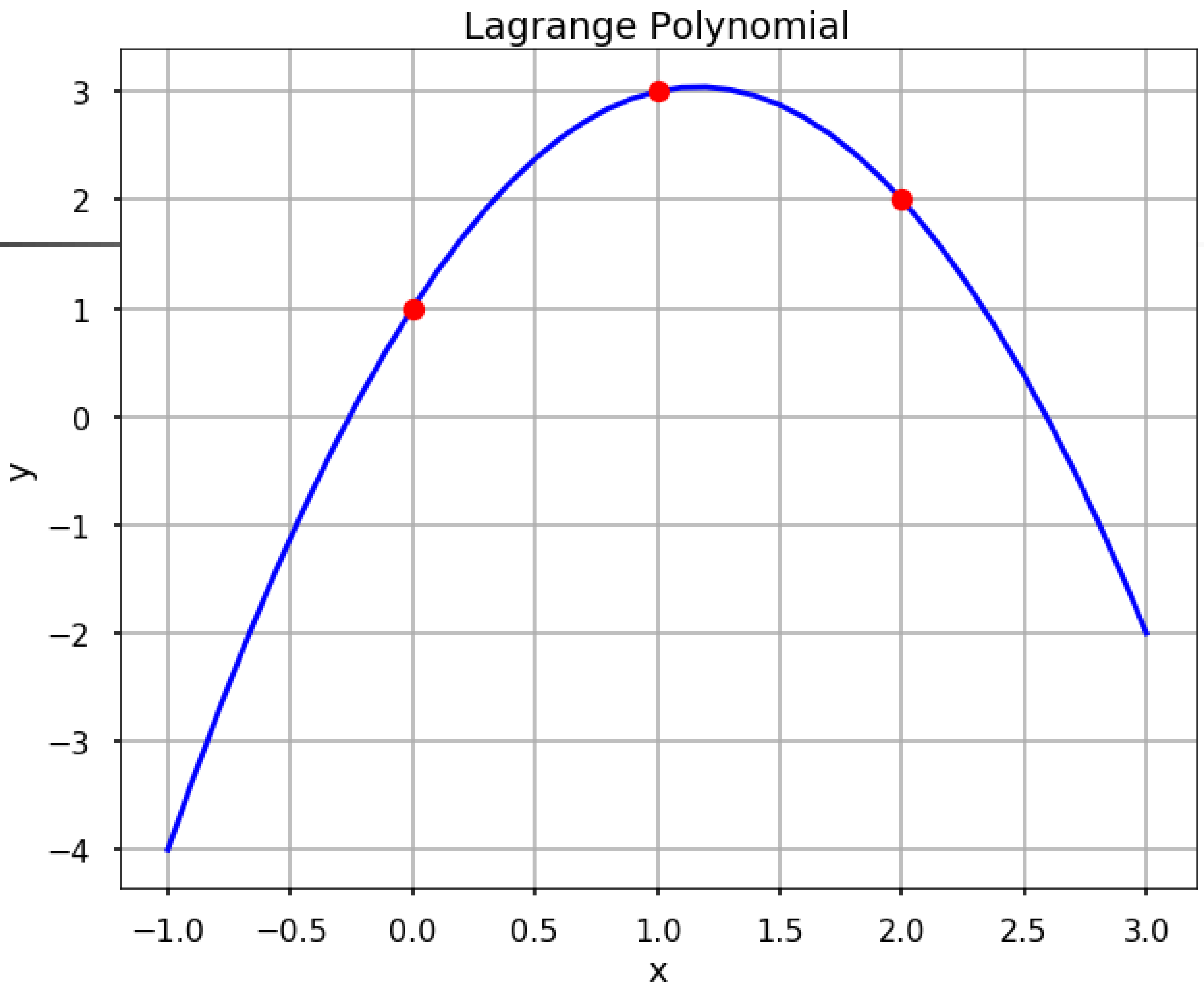
Example

- Find the Lagrange basis polynomials for the data set x =[0,1,2] and y= [1,3,2].
- Let's find $P_1(x)$, $P_2(x)$, and $P_3(x)$ accordingly.

Python illustration

```
import numpy as np
import numpy.polynomial.polynomial as poly
import matplotlib.pyplot as plt
x = [0, 1, 2]
y = [1, 3, 2]
P1_coeff = [1,-1.5,.5]
P2_coeff = [0, 2,-1]
P3_coeff = [0,-.5,.5]
```

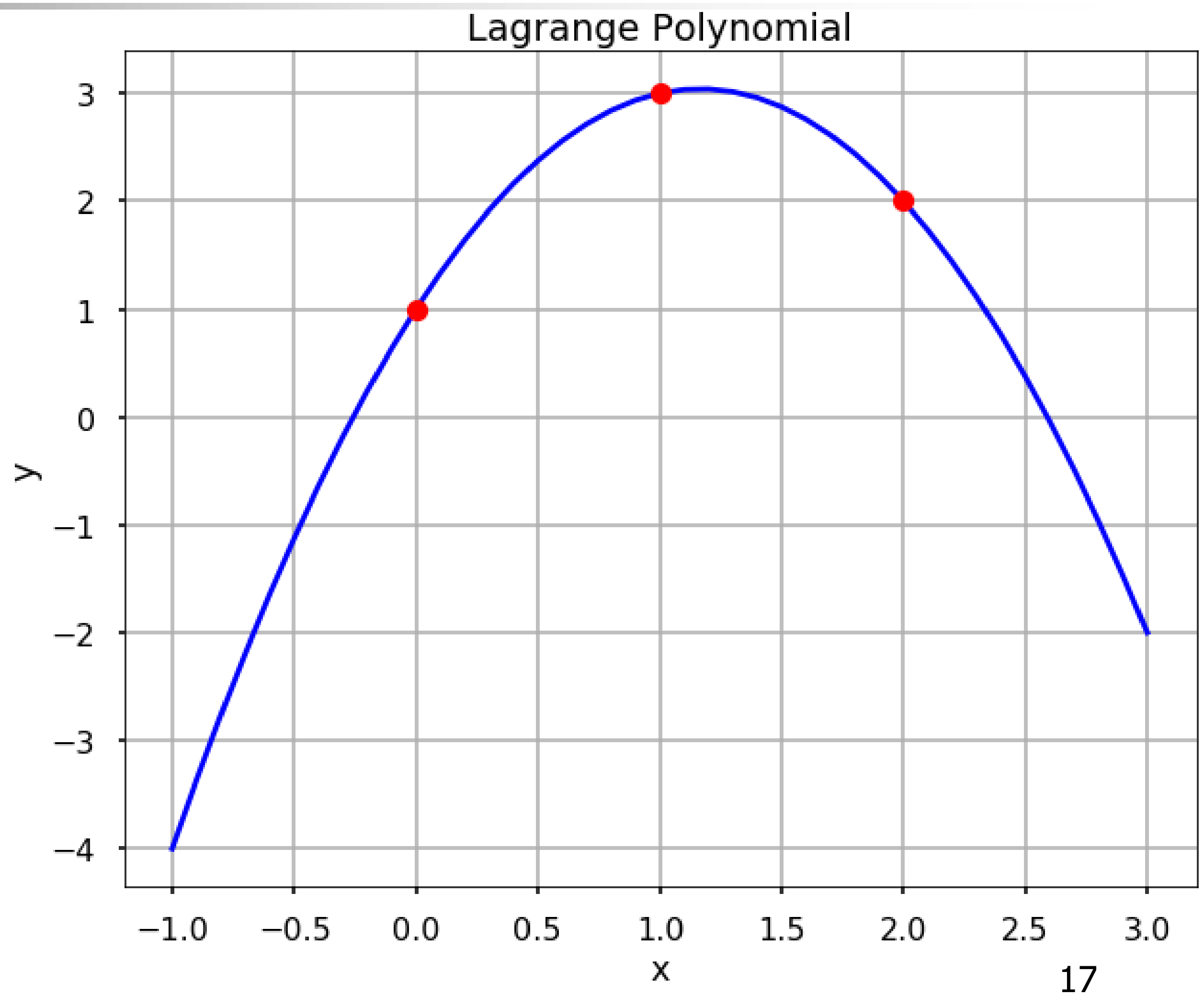
```
# get the polynomial function
P1 = poly.Polynomial(P1_coeff)
P2 = poly.Polynomial(P2_coeff)
P3 = poly.Polynomial(P3_coeff)
L = P1 + 3*P2 + 2*P3
x_new = np.arange(-1.0, 3.1, 0.1)
fig = plt.figure(figsize = (10,8))
plt.plot(x_new, L(x_new), 'b', x, y, 'ro')
```


Lagrange Polynomial
y
x
3
2
1
0
−1
−2
−3
−4
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Simpler implementation

- lagrange function in scipy.interpolate does everything for us

```
from scipy.interpolate import lagrange
f = lagrange(x, y)
fig = plt.figure(figsize = (10,8))
plt.plot(x_new, f(x_new), 'b', x, y, 'ro')
```



Newton's polynomial interpolation

- Use a n-1 order polynomial to go through n data points

$$f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots$$
$$+ a_n(x - x_0)(x - x_1)\ldots(x - x_n)$$

- Features: $f(x_i) = y_i$. Thus,

$$f(x_0) = a_0 = y_0$$

$$f(x_1) = a_0 + a_1(x_1 - x_0) = y_1$$

$$a_0 = y_0$$

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0}$$

$$a_2 = \frac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0}$$

Keep going

$$a_3 = \frac{\frac{\frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1}}{x_3-x_1} - \frac{\frac{y_2-y_1}{x_2-x_1} - \frac{y_1-y_0}{x_1-x_0}}{x_2-x_0}}{x_3-x_0}$$

- Define *divided differences*

$$f[x_1, x_0] = \frac{y_1-y_0}{x_1-x_0}$$

$$f[x_2, x_1, x_0] = \frac{\frac{y_2-y_1}{x_2-x_1} - \frac{y_1-y_0}{x_1-x_0}}{x_2-x_0} = \frac{f[x_2,x_1]-f[x_1,x_0]}{x_2-x_1}$$

$$f[x_k, x_{k-1}, \ldots, x_1, x_0] = \frac{f[x_k,x_{k-1},\ldots,x_2,x_1]-f[x_{k-1},x_{k-2},\ldots,x_1,x_0]}{x_k-x_0}$$

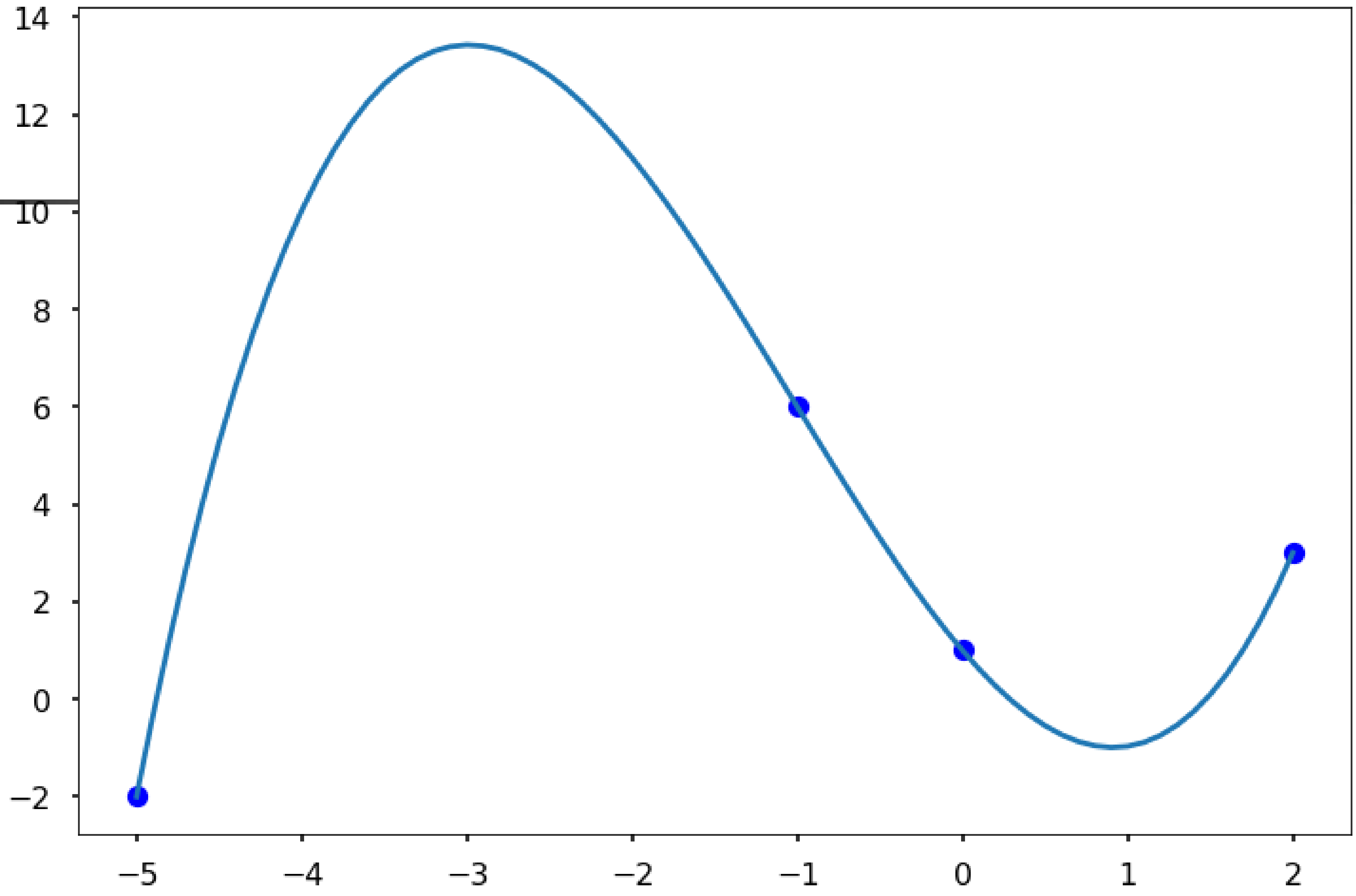
Calculate the coefficients

$$
\begin{array}{ccccc}
a_0 & a_1 & a_2 & a_3 & a_4 \\
y_0 & f[x_1,x_0] & f[x_2,x_1,x_0] & f[x_3,x_2,x_1,x_0] & f[x_4,x_3,x_2,x_1,x_0] \\
y_1 & f[x_2,x_1] & f[x_3,x_2,x_1] & f[x_4,x_3,x_2,x_1] & 0 \\
y_2 & f[x_3,x_2] & f[x_4,x_3,x_2] & 0 & 0 \\
y_3 & f[x_4,x_3] & 0 & 0 & 0 \\
y_4 & 0 & 0 & 0 & 0
\end{array}
$$

- Each element in the table can be calculated using the two previous elements (to the left).

Example

- Calculate the divided difference table for x =[-5, -1, 0, 2] and y=[-2,6,1,3]
- Create the divided_diff function based on the data to calculate the table and return the coefficients $a_0, \ldots, a_{n-1}$?
- Create the newton_poly function to evaluate the Newton's polynomial at a x* based on the given data and the calculated coefficients from divided_diff ?
- Plot your results


14
12
10
8
6
4
2
0
−2
−5
−4
−3
−2
−1
0
1
2