Programming and Computational Physics

# Interpolation

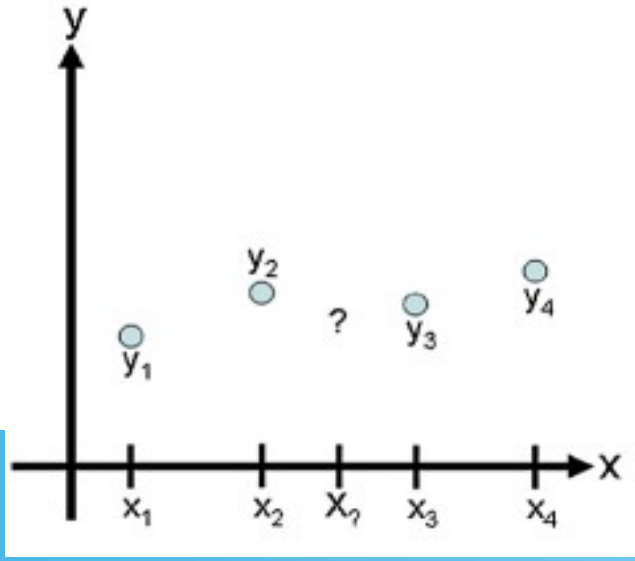
N. Petrov

# Content

- Interpolation Problem Statement
- Linear Interpolation
- Cubic Spline
- Lagrange Polynomial
- Newton’s Polynomial
- Tasks

# Interpolation Problem Statement

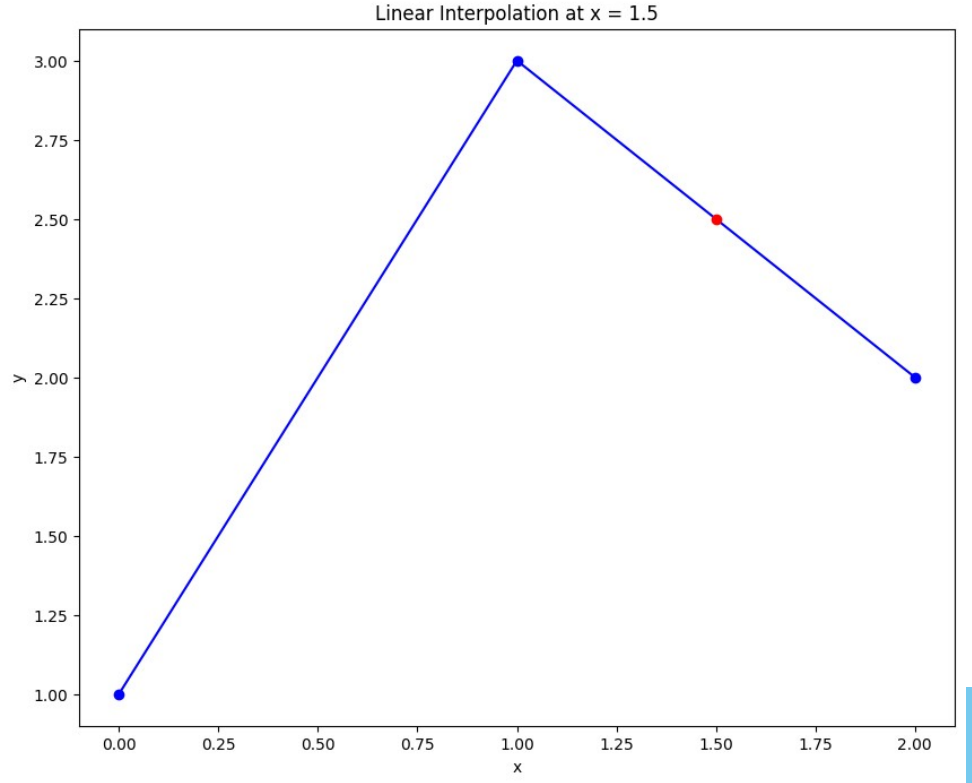
- Dataset consisting of independent data values $x_i$ and dependent data values $y_i$ for i = 1, …, n
- Need to find an estimation function Y such as $\hat{y}(x_i) = y_i$ for every point of dataset
- If given a new $x_k$ – the $\hat{y}(x_k)$ can be estimated
- $\hat{y}(x)$ – interpolation function
- We need to have underlying model for the data



# Linear Interpolation

- the estimated point is assumed to lie on the line joining the nearest points to the left and right
- If $x_i < x < x_{i+1}$

$$\hat{y}(x) = y_i + \frac{(y_{i+1} - y_i)(x - x_i)}{(x_{i+1} - x_i)}$$



# Cubic Spline

- the interpolation function is a set of piecewise cubic functions
- we assume that the points $(x_i, y_i)$ and $(x_{i+1}, y_{i+1})$ are joined by a cubic polynomial, valid for $x_i \le x \le x_{i+1}$ for $i = 1, \ldots, n-1$:

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

- To find the interpolation function, we must first determine the coefficients $a_i$, $b_i$, $c_i$, $d_i$ for each of the cubic functions
- For n points, there are $n-1$ cubic functions to find, and each cubic function requires four coefficients (total of $4(n-1)$ unknowns / equations)

# Cubic Spline

- First, the cubic functions must intersect the data the points on the left and the right (2(n − 1) equations):

$$S_i(x_i) = y_i, \qquad i = 1, \ldots, n-1,$$

$$S_i(x_{i+1}) = y_{i+1}, \qquad i = 1, \ldots, n-1,$$

- Next, we want each cubic function to join as smoothly with its neighbors as possible, so we constrain the splines to have continuous first and second derivatives at the data points

  i = 2, . . . , n − 1 (2(n − 2) equations):

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}), \quad i = 1, \ldots, n-2,$$

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}), \quad i = 1, \ldots, n-2,$$

# Cubic Spline

- Final two equations are arbitrary – to fit the circumstances of the interpolation being performed (boundary conditions)
- If the curve is a straight line at the endpoints:

$$S_1''(x_1) = 0,$$
$$S_{n-1}''(x_n) = 0.$$

# Cubic Spline

- constraints $S_i(x_i) = y_i$:

$$a_1x_1^3 + b_1x_1^2 + c_1x_1 + d_1 = y_1,$$
$$a_2x_2^3 + b_2x_2^2 + c_2x_2 + d_2 = y_2,$$
$$\dots$$
$$a_{n-1}x_{n-1}^3 + b_{n-1}x_{n-1}^2 + c_{n-1}x_{n-1} + d_{n-1} = y_{n-1}.$$

- constraints $S_i(x_{i+1}) = y_{i+1}$:

$$a_1x_2^3 + b_1x_2^2 + c_1x_2 + d_1 = y_2,$$
$$a_2x_3^3 + b_2x_3^2 + c_2x_3 + d_2 = y_3,$$
$$\dots$$
$$a_{n-1}x_n^3 + b_{n-1}x_n^2 + c_{n-1}x_n + d_{n-1} = y_n.$$

# Cubic Spline

- constraints $S'_i(x_{i+1}) = S'_{i+1}(x_{i+1})$:

$$3a_1x_2^2 + 2b_1x_2 + c_1 - 3a_2x_2^2 - 2b_2x_2 - c_2 = 0,$$
$$3a_2x_3^2 + 2b_2x_3 + c_2 - 3a_3x_3^2 - 2b_3x_3 - c_3 = 0,$$
$$\dots$$
$$3a_{n-2}x_{n-1}^2 + 2b_{n-2}x_{n-1} + c_{n-2} - 3a_{n-1}x_{n-1}^2 - 2b_{n-1}x_{n-1} - c_{n-1} = 0.$$

- constraints $S''_i(x_{i+1}) = S''_{i+1}(x_{i+1})$:

$$6a_1x_2 + 2b_1 - 6a_2x_2 - 2b_2 = 0,$$
$$6a_2x_3 + 2b_2 - 6a_3x_3 - 2b_3 = 0,$$
$$\dots$$
$$6a_{n-2}x_{n-1} + 2b_{n-2} - 6a_{n-1}x_{n-1} - 2b_{n-1} = 0.$$

# Cubic Spline

- Boundary conditions $S''_1(x_1) = 0$ and $S''_{n-1}(x_n) = 0$:

$$6a_1x_1 + 2b_1 = 0,$$
$$6a_{n-1}x_n + 2b_{n-1} = 0.$$

- System of equations:

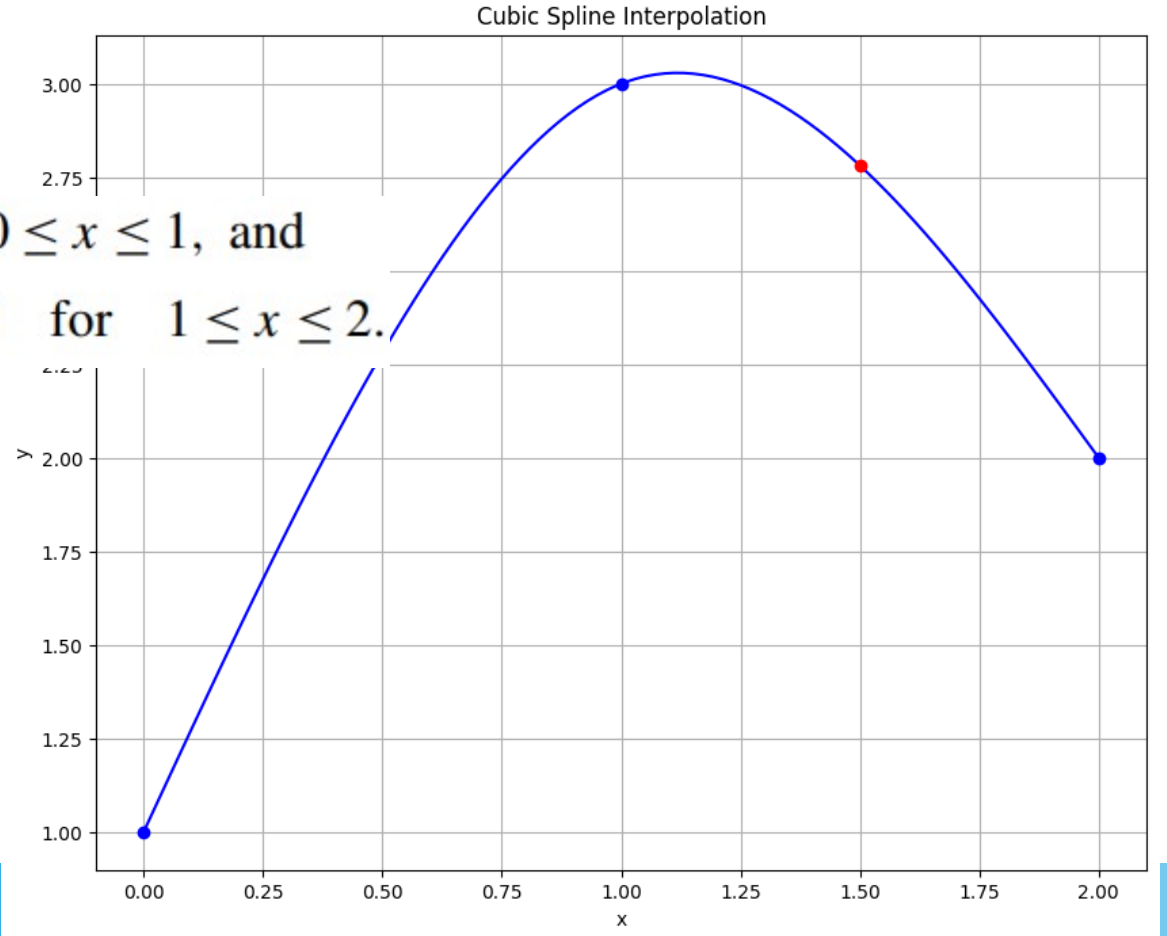
$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 4 & 2 & 1 \\ 3 & 2 & 1 & 0 & -3 & -2 & -1 & 0 \\ 6 & 2 & 0 & 0 & -6 & -2 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 3 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

# Cubic Spline

- two cubic polynomials:

$$S_1(x) = -0.75x^3 + 2.75x + 1 \quad \text{for} \quad 0 \le x \le 1, \text{ and}$$

$$S_2(x) = 0.75x^3 - 4.5x^2 + 7.25x - 0.5 \quad \text{for} \quad 1 \le x \le 2.$$



# Lagrange Polynomial Interpolation

- Lagrange polynomial interpolation finds a single polynomial that goes through all the data points L(x)
- As an interpolation function, it should have the property $L(x_i) = y_i$
- Lagrange polynomials can be represented as a linear combination of Lagrange basis polynomials $P_i(x)$:

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

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

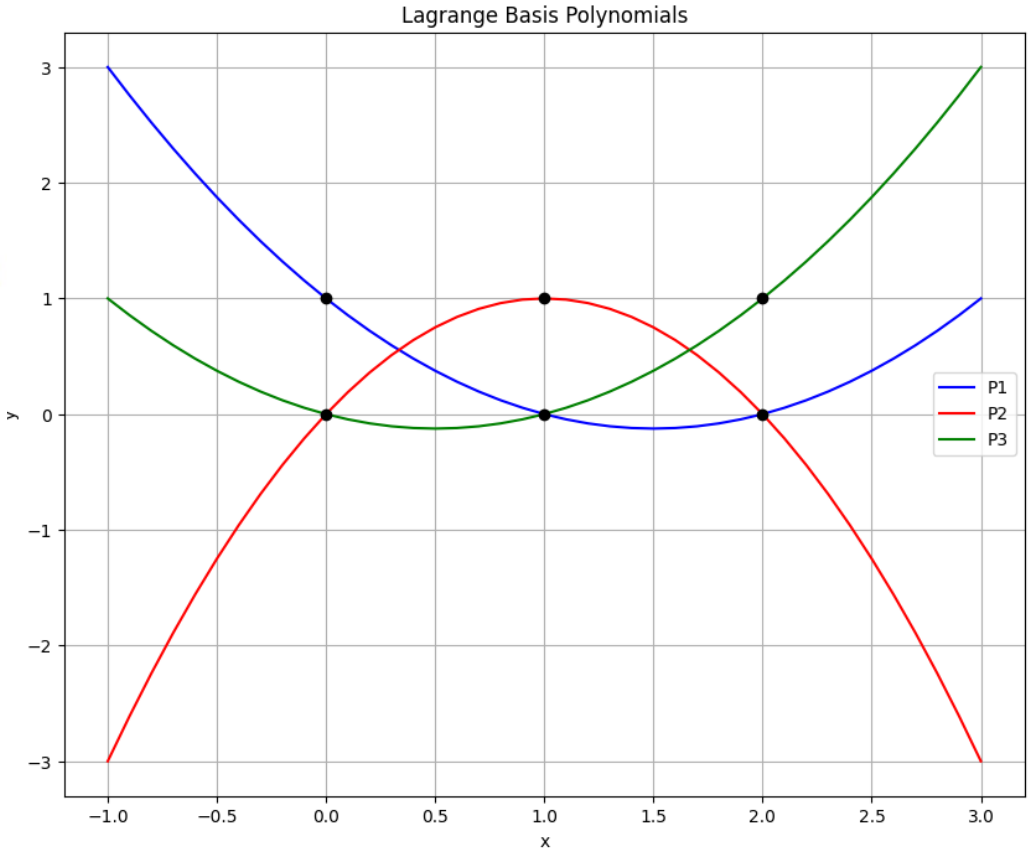
# Lagrange base polynomials

For x = [0, 1, 2]

$$P_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} = \frac{1}{2}(x^2 - 3x + 2),$$

$$P_2(x) = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} = -x^2 + 2x,$$

$$P_3(x) = \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} = \frac{1}{2}(x^2 - x).$$

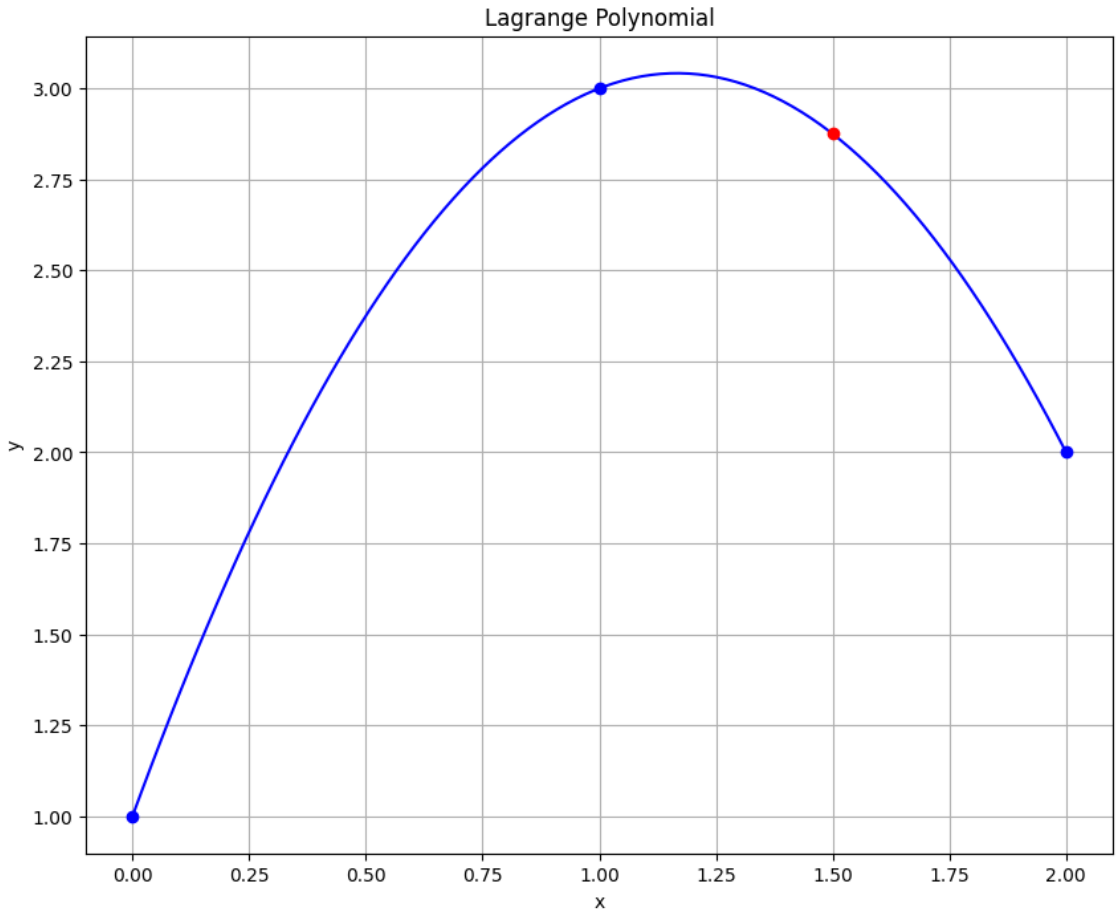


# Lagrange Polynomial Interpolation

by construction, $P_i(x)$ has the property that:

$P_i(x_k) = 1$ when $i = k$ and

$P_i(x_k) = 0$ when $i \neq k$



# Newton's Polynomial Interpolation

- Newton's polynomial interpolation is another popular way to exactly fit a set of data points

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

which can be rewritten as:

$$f(x) = \sum_{i=0}^{n} a_i n_i(x)$$

where

$$n_i(x) = \prod_{j=0}^{i-1} (x - x_j)$$

# Newton's Polynomial Interpolation

- Feature of this approach – coefficients $a_i$ can be determined using very simple mathematical procedure
- since the polynomial goes through each data point, for a data point $(x_i, y_i)$, we will have $f(x_i) = y_i$, thus we have:

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

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

Then we can find coefficient $a_1$:

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

# Newton's Polynomial

- If we insert data point ($x_2$, $y_2$), we can calculate $a_2$:

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

- One more point:

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

- The patterns is divided differences:

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

# Newton's Polynomial

- Then:

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

- We could obtain the following iteration equation:

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

# Newton's Polynomial

- The procedure for finding these coefficients can be summarized into a divided difference table:

| | | | | | |
|---|---|---|---|---|---|
| $x_0$ | $y_0$ | | | | |
| | | $f[x_1, x_0]$ | | | |
| $x_1$ | $y_1$ | | $f[x_2, x_1, x_0]$ | | |
| | | $f[x_2, x_1]$ | | $f[x_3, x_2, x_1, x_0]$ | |
| $x_2$ | $y_2$ | | $f[x_3, x_2, x_1]$ | | $f[x_4, x_3, x_2, x_1, x_0]$ |
| | | $f[x_3, x_2]$ | | $f[x_4, x_3, x_2, x_1]$ | |
| $x_3$ | $y_3$ | | $f[x_4, x_3, x_2]$ | | |
| | | $f[x_4, x_3]$ | | | |
| $x_4$ | $y_4$ | | | | |

# Newton's Polynomial

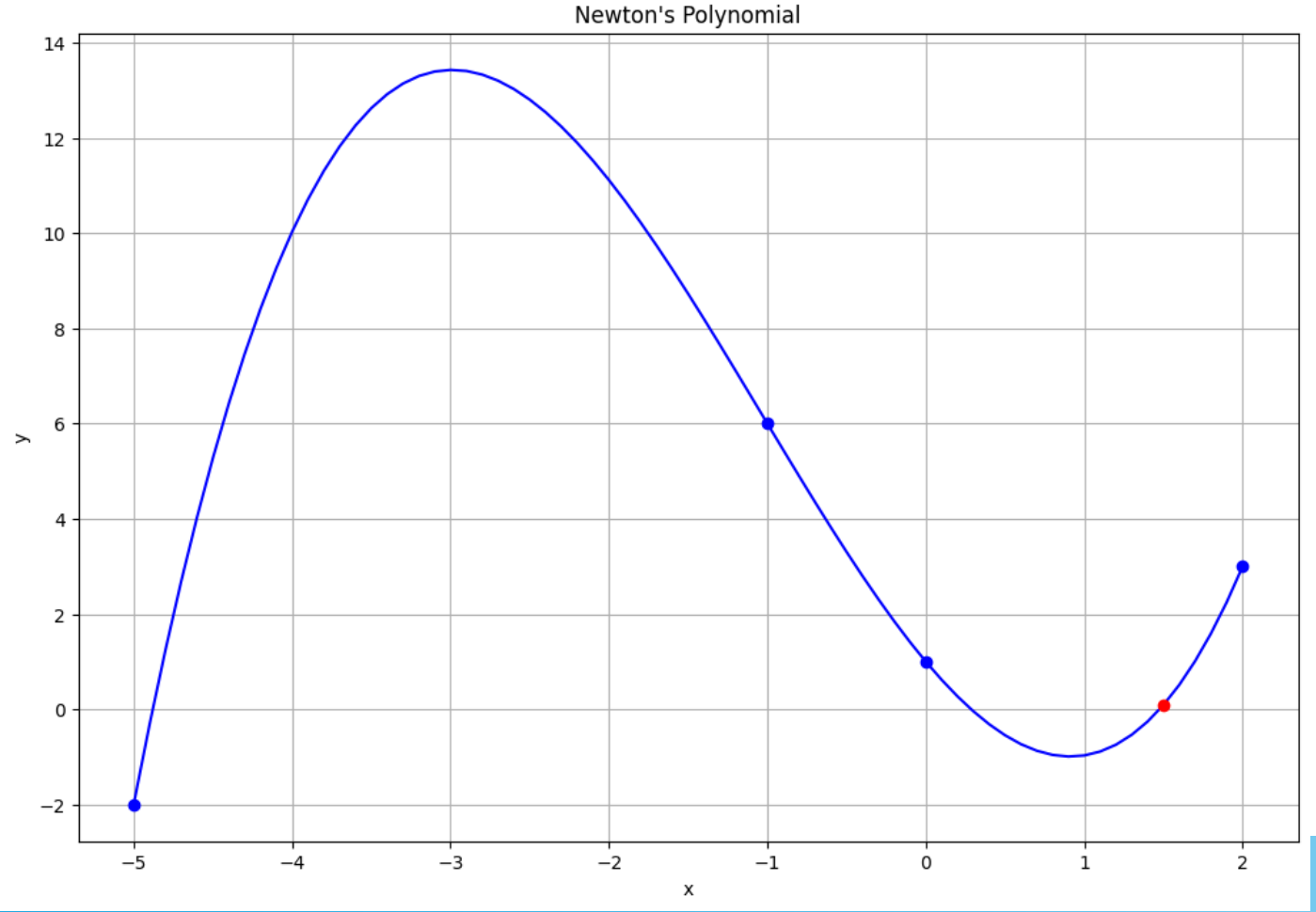
- we can calculate all elements and store them in a diagonal matrix

| | | | | |
|---|---|---|---|---|
| $y_0$ | $f[x_1, x_0]$ | $f[x_2, x_1, x_0]$ | $f[x_3, x_2, x_1, x_0]$ | $f[x_4, x_3, x_2, x_1, x_0]$ |
| $y_1$ | $f[x_2, x_1]$ | $f[x_3, x_2, x_1]$ | $f[x_4, x_3, x_2, x_1]$ | 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 |

- The first row in the matrix are the coefficients that we need ($a_1$, $a_2$, $a_3$, $a_4$)

# Newton’s Polynomial

For x = [-5, -1, 0, 2]
y = [-2, 6, 1, 3]



# Complexity

- Lagrange polynomial – $O(n^2)$
- Newton’s polynomial – $O(n)$

# Tasks

Given points with: x = [0, 1, 2], y = [1, 3, 2]

- Implement linear interpolation for $x_k = 1.5$ and plot the result
- Check solution if using *interp1d* from scipy.interpolate
- Implement cubic spline interpolation for the same point and plot results
- Check solution if using *CubicSpline* from scipy.interpolate
- Find Lagrange base polynomials (P1, P2 and P3) for the same points. Plot them in for x = np.arange(-1, 3.1, 0.1)
- Find L(x), and $x_k$ , plot results and compare to *lagrange* from scipy.interpolate
- Play with number of the dots and observe behavior of the polynomial
- For points: x = [-5,-1,0,2], y = [-2,6,1,3] find newton's polynomial and get $y_k$ = ? for $x_k = 1.5$

# References


- Python Programming and Numerical Methods - A Guide for Engineers and Scientists ISBN: 9780128195499