Review of Finite Difference Approximation

Types of Finite Differences

Finite difference approximation is a numerical method used to approximate derivatives.

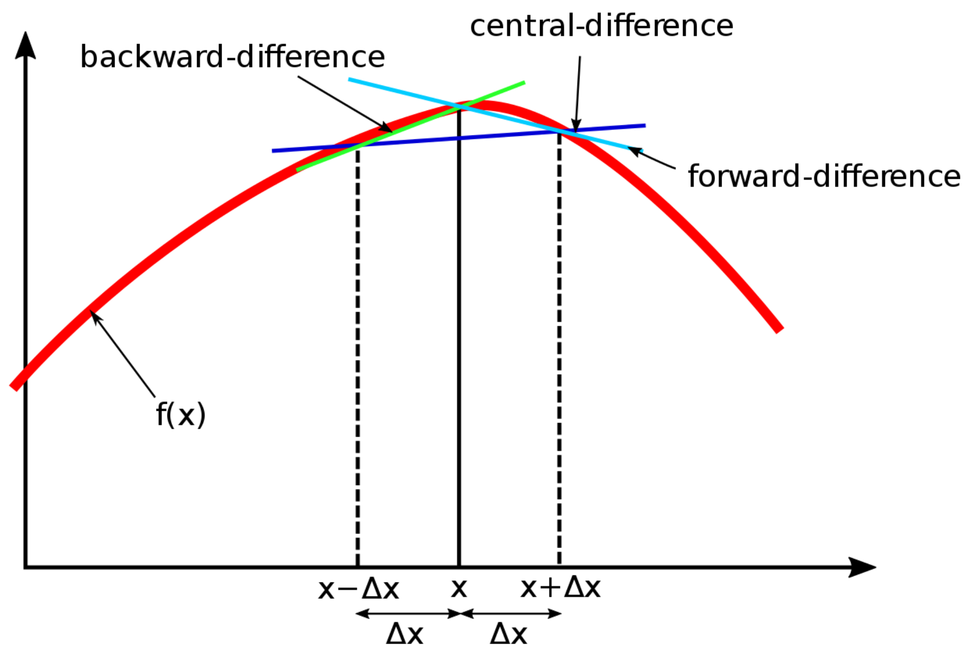
For a function $f(x)$, the first derivative can be approximated as:

- Forward Difference: $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
- Backward Difference: $f'(x) \approx \frac{f(x)-f(x-h)}{h}$
- Central Difference: $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$

Here, h is the step size.

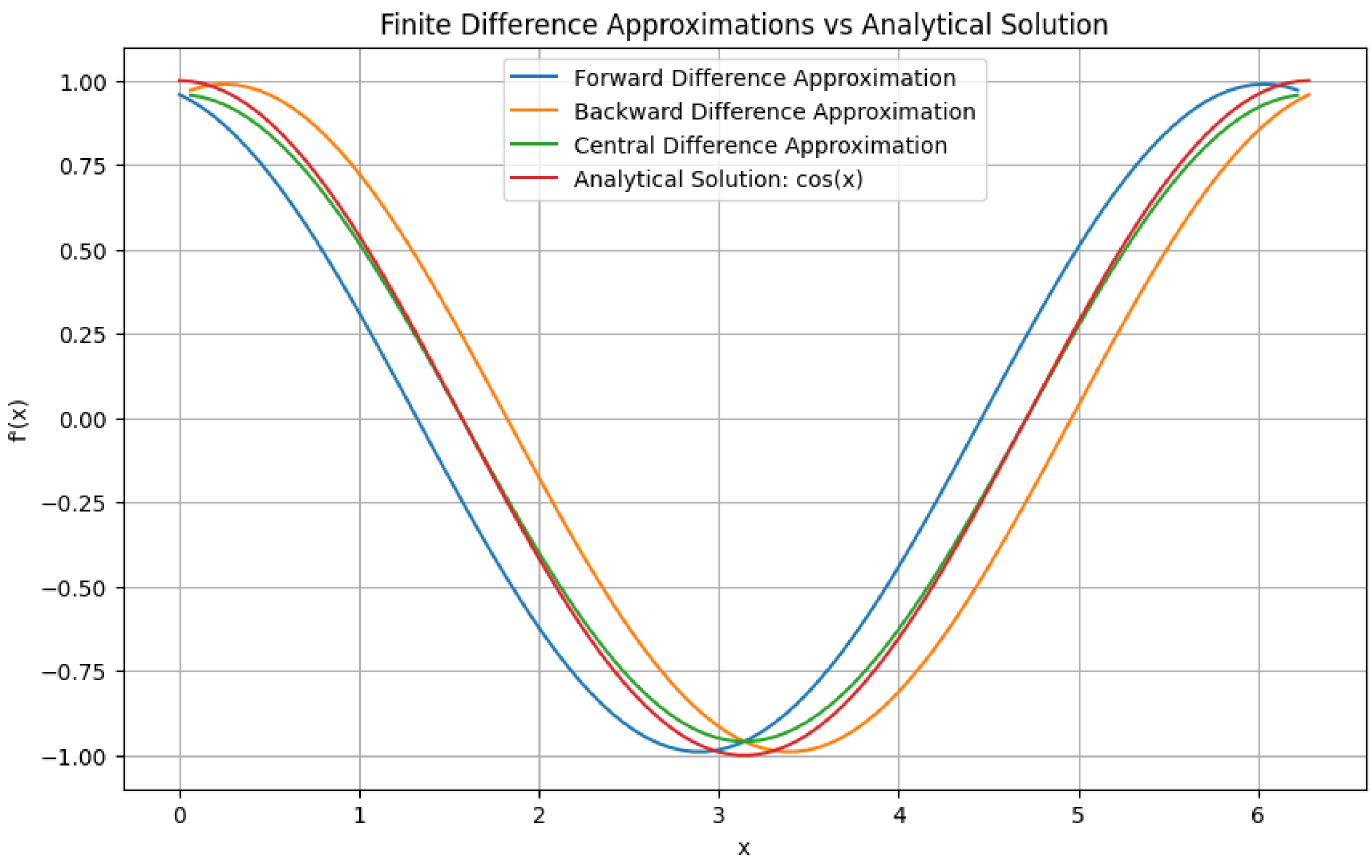
For the second derivative f'', we also have a central finite difference approximation:

$$f''(x) \approx \frac{1}{h^2}(f(x+h) - 2f(x) + f(x-h))$$



In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return np.sin(x)

def forward_difference(f, x, h):
    return (f(x + h) - f(x)) / h

def backward_difference(f, x, h):
    return (f(x) - f(x - h)) / h

def central_difference(f, x, h):
    return (f(x + h) - f(x - h)) / (2 * h)
```

```python
x_values = np.linspace(0, 2*np.pi, 100)
y_values = f(x_values)

h = 0.5

derivatives_fd = [forward_difference(f, x, h) for x in x_values[:-1]]
derivatives_bd = [backward_difference(f, x, h) for x in x_values[1:]]
derivatives_cd = [central_difference(f, x, h) for x in x_values[1:-1]]

plt.figure(figsize=(10, 6))
plt.plot(x_values[:-1], derivatives_fd, label='Forward Difference Approximation')
plt.plot(x_values[1:], derivatives_bd, label='Backward Difference Approximation')
plt.plot(x_values[1:-1], derivatives_cd, label='Central Difference Approximation')
plt.plot(x_values, np.cos(x_values), label='Analytical Solution: cos(x)')
plt.title('Finite Difference Approximations vs Analytical Solution')
plt.xlabel('x')
plt.ylabel('f\'(x)')
plt.legend()
plt.grid(True)
plt.show()
```



Error Analysis

Recall: Given that $f(x) \in C^\infty$ is a smooth function. Its Taylor expansion about the point $x = c$ is:

$$
\begin{aligned}
f(x) &= f(c) + f'(c)(x-c) + \frac{1}{2!}f''(c)(x-c)^2 + \frac{1}{3!}f'''(c)(x-c)^3 + \cdots \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} f^{(k)}(c)(x-c)^k
\end{aligned}
$$

The Taylor expansion for $f(x+h)$ about x :

$$f(x+h)=\sum_{k=0}^{\infty}\frac{1}{k!}f^{(k)}(x)h^k=\sum_{k=0}^{n}\frac{1}{k!}f^{(k)}(x)h^k+E_{n+1}$$

where

$$E_{n+1}=\sum_{k=n+1}^{\infty}\frac{1}{k!}f^{(k)}(x)h^k=\frac{1}{(n+1)!}f^{(n+1)}(\xi)h^{n+1}=\mathcal{O}\left(h^{n+1}\right)$$

$$\begin{aligned}
f(x+h)&=f(x)+hf'(x)+\frac{1}{2}h^2f''(x)+\frac{1}{6}h^3f'''(x)+\mathcal{O}\left(h^4\right),\\
f(x-h)&=f(x)-hf'(x)+\frac{1}{2}h^2f''(x)-\frac{1}{6}h^3f'''(x)+\mathcal{O}\left(h^4\right).
\end{aligned}$$

Forward Euler:

$$\frac{f(x+h)-f(x)}{h}=f'(x)+\frac{1}{2}hf''(x)+\mathcal{O}\left(h^2\right)=f'(x)+\mathcal{O}\left(h^1\right),\quad \left(1^{\text{st}}\ \text{order}\right),$$

Backward Euler:

$$\frac{f(x)-f(x-h)}{h}=f'(x)-\frac{1}{2}hf''(x)+\mathcal{O}\left(h^2\right)=f'(x)+\mathcal{O}\left(h^1\right),\quad \left(1^{\text{st}}\ \text{order}\right),$$

Central finite difference:

$$\frac{f(x+h)-f(x-h)}{2h}=f'(x)-\frac{1}{6}h^2f'''(x)+\mathcal{O}\left(h^2\right)=f'(x)+\mathcal{O}\left(h^2\right),\quad \left(2^{\text{nd}}\ \text{order}\right),$$

Polynomial Interpolation

We aim to interpolate a dataset using a polynomial.

Problem Description:

Given $(n+1)$ points, denoted as (x_i, y_i) where $i=0,1,2,\ldots,n$, with distinct x_i (not necessarily sorted), we seek to find a polynomial of degree n,

$$P_n(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$$

such that it interpolates these points, i.e.,

$$P_n(x_i)=y_i\quad \text{for } i=0,1,2,\ldots,n.$$

Reasons for Polynomial Interpolation:

- Determine intermediate values within a discrete dataset.
- Approximate a potentially complex function with a polynomial.
- Simplify calculations such as derivatives and integrals.

Example: Consider the following dataset:

x_i	-1	0	1
y_i	0	1	1

Solution:

Let $P(x) = a_2x^2 + a_1x + a_0$ be the polynomial we want to find.

We use the interpolation conditions to determine the coefficients a_2, a_1, and a_0:

1. For $x = -1, y = 0$:

$$P(-1) = a_2(-1)^2 + a_1(-1) + a_0 = 0$$

$$a_2 - a_1 + a_0 = 0$$

2. For $x = 0, y = 1$:

$$P(0) = a_0 = 1$$

3. For $x = 1, y = 1$:

$$P(1) = a_2(1)^2 + a_1(1) + a_0 = 1$$

$$a_2 + a_1 + a_0 = 1$$

Now, we solve these equations:

- From $a_0 = 1$, substitute into the other equations:

$$a_2 - a_1 + 1 = 0$$

$$a_2 + a_1 + 1 = 1$$

- Subtract the first equation from the second:

$$2a_1 = 1$$

$$a_1 = \frac{1}{2}$$

- Substitute $a_1 = \frac{1}{2}$ back into $a_2 - \frac{1}{2} + 1 = 0$:

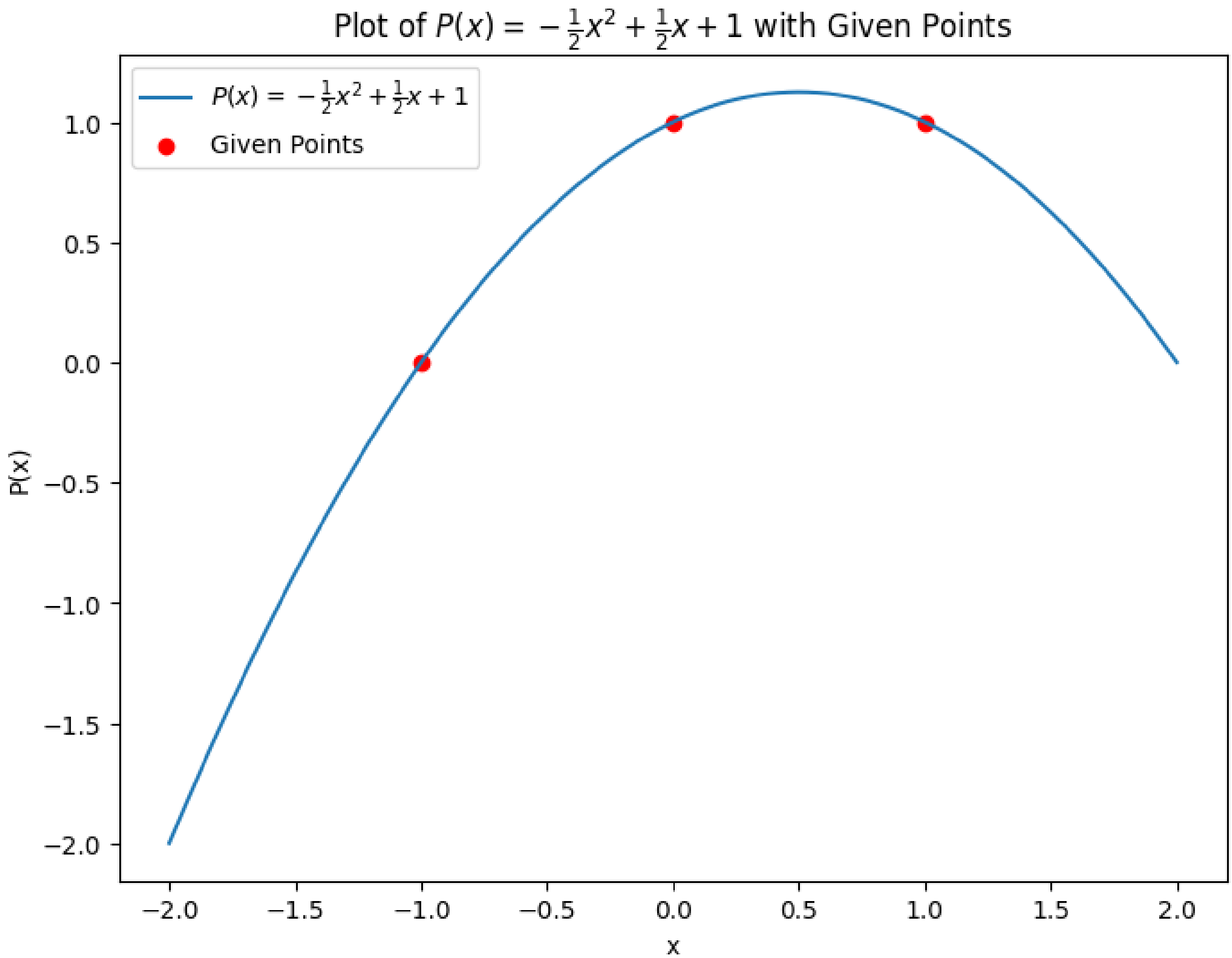
$$a_2 = \frac{1}{2} - 1 = -\frac{1}{2}$$

Therefore, the polynomial $P(x)$ that interpolates the given dataset is:

$$P(x) = -\frac{1}{2}x^2 + \frac{1}{2}x + 1$$

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def P(x):
    return -0.5 * x**2 + 0.5 * x + 1

x_points = np.array([-1, 0, 1])
y_points = np.array([0, 1, 1])

x_values = np.linspace(-2, 2, 400)
```

```python
y_values = P(x_values)

plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, label='$P(x) = -\\frac{1}{2} x^2 + \\frac{1}{2} x + 1$')
plt.scatter(x_points, y_points, color='red', label='Given Points')
plt.title('Plot of $P(x) = -\\frac{1}{2} x^2 + \\frac{1}{2} x + 1$ with Given Points')
plt.xlabel('x')
plt.ylabel('P(x)')
plt.legend()
plt.show()
```



The above equations can be written in matrix form: $$

$$\begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

\begin{bmatrix} a_0 \ a_1 \ a_2

\end{bmatrix}

$$\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$

For the general case with $(n+1)$ points, we have

$$P_n(x_i) = y_i, \quad i = 0, 1, 2, \cdots, n$$

We will have $(n+1)$ equations and $(n+1)$ unknowns:

$$\begin{array}{rcl}
P_n(x_0) = y_0 & : & x_0^n a_n + x_0^{n-1} a_{n-1} + \cdots + x_0 a_1 + a_0 = y_0 \\
P_n(x_1) = y_1 & : & x_1^n a_n + x_1^{n-1} a_{n-1} + \cdots + x_1 a_1 + a_0 = y_1 \\
\vdots & & \\
P_n(x_n) = y_n & : & x_n^n a_n + x_n^{n-1} a_{n-1} + \cdots + x_n a_1 + a_0 = y_n
\end{array}$$

In matrix-vector form

$$\begin{pmatrix} x_0^n & x_0^{n-1} & \ldots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \ldots & x_1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^n & x_n^{n-1} & \ldots & x_n & 1 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

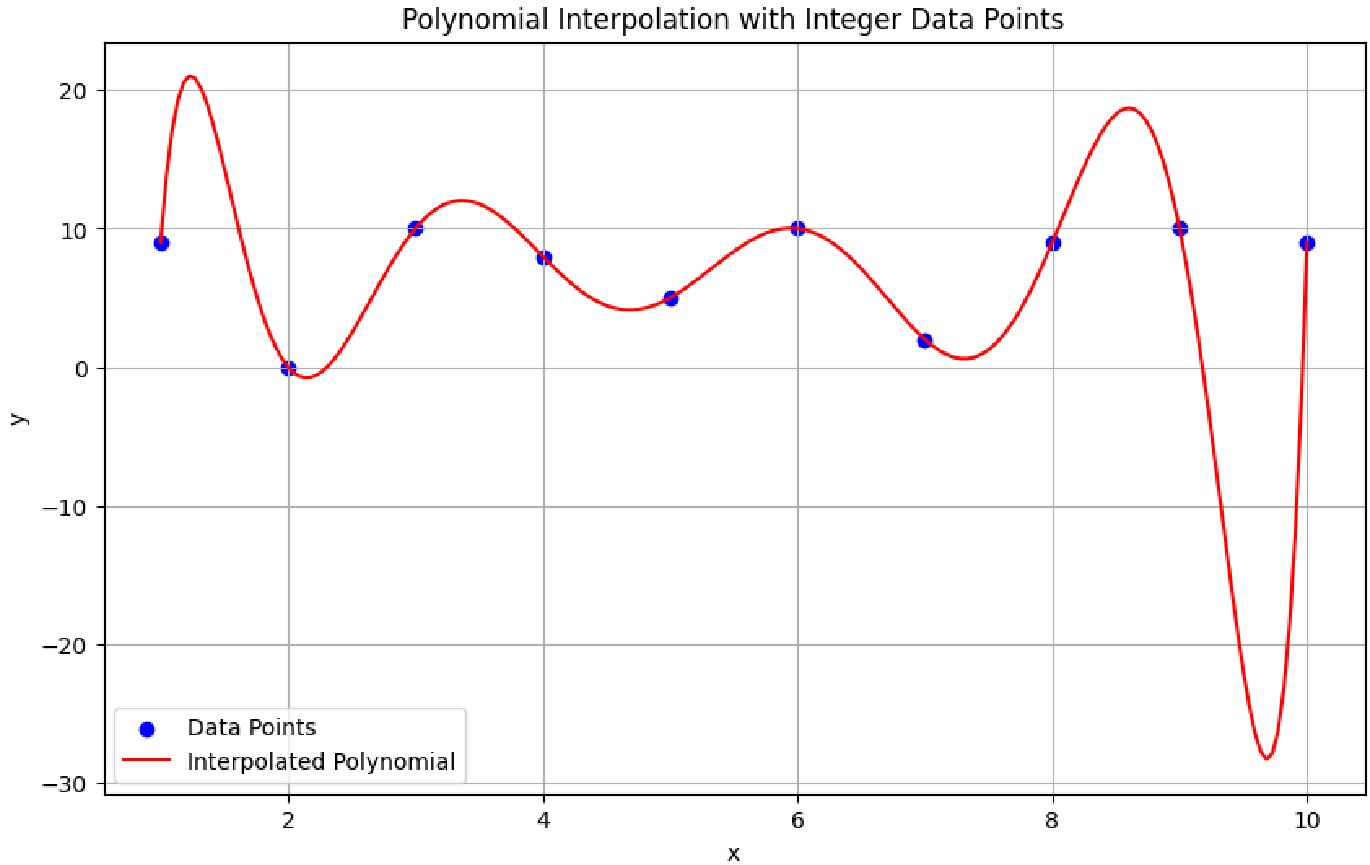
or

$$\mathbf{X} a = y$$

- $\mathbf{X}$: $(n+1) \times (n+1)$ matrix (the van cer Monde matrix) given by x_i
- a: unknown vector, with length $(n+1)$
- y: length $(n+1)$ vector given by y_i

In [ ]:

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

np.random.seed(326)
x = np.arange(1, 11)
y = np.random.randint(0, 11, size=10)

degree = len(x) - 1  # Degree of polynomial to fit
A = np.vander(x, degree + 1, increasing=False)  # Vandermonde matrix
coefficients = np.linalg.solve(A, y)  # Solve system of equations to find coefficients

def poly_function(coefficients):
    def func(x):
        return sum(c * x**i for i, c in enumerate(coefficients[::-1]))
    return func

x_values = np.linspace(1, 10, 200)
y_values = poly_function(coefficients)(x_values)

plt.figure(figsize=(10, 6))
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x_values, y_values, color='red', label='Interpolated Polynomial')
plt.title('Polynomial Interpolation with Integer Data Points')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```



Lagrange Form

Given points: $x_0, x_1, \cdots, x_n$.

Define the cardinal functions $l_0, l_1, \cdots, l_n \in \mathcal{P}^n$, satisfying the properties

$$l_i\left(x_j\right) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad i = 0, 1, \cdots, n$$

where δ_{ij} is the Kronecker's delta and $\mathcal{P}^n$ is the set of polynomials up to degree n.

The cardinal functions $l_i(x)$ can be written as

$$\begin{aligned} l_i(x) &= \prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j}\right) \\ &= \frac{x - x_0}{x_i - x_0} \cdot \frac{x - x_1}{x_i - x_1} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_n}{x_i - x_n} \end{aligned}$$

Lagrange form of the interpolation polynomial can be simply expressed as

$$P_n(x) = \sum_{i=0}^{n} l_i(x) \cdot y_i.$$

Example

Consider the following dataset:

x_i	-1	0	1
y_i	0	1	1

The Lagrange interpolation polynomial $P(x)$ is given by:

$$P(x) = \sum_{i=0}^{n} y_i \cdot \ell_i(x)$$

where $\ell_i(x)$ are the Lagrange basis polynomials:

$$\ell_i(x) = \prod_{\substack{0 \le j \le n \\ j \ne i}} \frac{x - x_j}{x_i - x_j}$$

Let's compute $P(x)$ for the given dataset:

1. **Compute the Lagrange basis polynomials** $\ell_i(x)$:

   For $i = 0$:

   $$\ell_0(x) = \frac{(x-0)(x-1)}{(-1-0)(-1-1)} = \frac{x(x-1)}{2}$$

   For $i = 1$:

   $$\ell_1(x) = \frac{(x+1)(x-1)}{(0+1)(0-1)} = -(x+1)(x-1)$$

   For $i = 2$:

   $$\ell_2(x) = \frac{(x+1)(x-0)}{(1+1)(1-0)} = \frac{x(x+1)}{2}$$

2. **Construct the Lagrange interpolation polynomial** $P(x)$:

   $$P(x) = 0 \cdot \ell_0(x) + 1 \cdot \ell_1(x) + 1 \cdot \ell_2(x)$$

   $$P(x) = -(x+1)(x-1) + \frac{x(x+1)}{2}$$

   $$P(x) = -\frac{x^2}{2} + \frac{x}{2} + 1$$

Newton's Divided Differences

Given a data set

x_i	x_0	x_1	$\cdots$	x_n
y_i	y_0	y_1	$\cdots$	y_n

Main idea: Starting with $P_k(x)$, which interpolates $k+1$ data points $\{x_i, y_i\}$ for $i = 0, 1, 2, \ldots, k$, find $P_{k+1}(x)$ that incorporates an additional point $\{x_{k+1}, y_{k+1}\}$ by utilizing P_k and adding an extra term.

- For $n=0$, $P_0(x)=y_0$.
- For $n=1$, $P_1(x)=P_0(x)+a_1\left(x-x_0\right)$, where we need to find a_1 that satisfies the interpolating requirement:

$$y_1=P_1\left(x_1\right)=P_0\left(x_1\right)+a_1\left(x_1-x_0\right)=y_0+a_1\left(x_1-x_0\right),$$

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

- For $n=2$, $P_2(x)=P_1(x)+a_2\left(x-x_0\right)\left(x-x_1\right)$, where we need to find a_2 that satisfies the interpolating requirement:

$$y_2=P_2\left(x_2\right)=P_1\left(x_2\right)+a_2\left(x_2-x_0\right)\left(x_2-x_1\right),$$

$$a_2=\frac{y_2-P_1\left(x_2\right)}{\left(x_2-x_0\right)\left(x_2-x_1\right)}.$$

Verify that P_0, P_1, and P_2 satisfies the interpolating requirement on your own.

It is inconvinient to have P_1 in the formula for a_2. We would like to express a_2 in a different way. Recall

$$P_1(x)=y_0+\frac{y_1-y_0}{x_1-x_0}\left(x-x_0\right).$$

Then

$$\begin{aligned}
P_1\left(x_2\right)&=y_0+\frac{y_1-y_0}{x_1-x_0}\left(x_2-x_0\right)\\
&=y_0+\frac{y_1-y_0}{x_1-x_0}\left(x_2-x_1\right)+\frac{y_1-y_0}{x_1-x_0}\left(x_1-x_0\right)\\
&=y_1+\frac{y_1-y_0}{x_1-x_0}\left(x_2-x_1\right).
\end{aligned}$$

Then, a_2 can be rewritten as

$$a_2=\frac{y_2-P_1\left(x_2\right)}{\left(x_2-x_0\right)\left(x_2-x_1\right)}=\frac{y_2-y_1-\frac{y_1-y_0}{x_1-x_0}\left(x_2-x_1\right)}{\left(x_2-x_0\right)\left(x_2-x_1\right)}=\frac{\frac{y_2-y_1}{x_2-x_1}-\frac{y_1-y_0}{x_1-x_0}}{x_2-x_0}.$$

The **general case** for a_n : Assume that $P_{n-1}(x)$ interpolates $\left(x_i, y_i\right)$ for $i=0,1,\cdots,n-1$. Let

$$P_n(x)=P_{n-1}(x)+a_n\left(x-x_0\right)\left(x-x_1\right)\cdots\left(x-x_{n-1}\right)$$

Then for $i=0,1,\cdots,n-1$, we have

$$P_n\left(x_i\right)=P_{n-1}\left(x_i\right)=y_i.$$

Find a_n by the property $P_n\left(x_n\right)=y_n$,

$$y_n=P_{n-1}\left(x_n\right)+a_n\left(x_n-x_0\right)\left(x_n-x_1\right)\cdots\left(x_n-x_{n-1}\right)$$

then

$$a_n=\frac{y_n-P_{n-1}\left(x_n\right)}{\left(x_n-x_0\right)\left(x_n-x_1\right)\cdots\left(x_n-x_{n-1}\right)}$$

Newton's form for the interpolation polynomial:

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

The number of floating point operations needed to evaluate this polynomial is
$1 + 3 + 5 + 7 \ldots + 2n + 1 = (n+1)^2 = O(n^2)$.

Nested form of Newton's polynomial:

$$\begin{aligned} P_n(x) =& a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \\ =& a_0 + (x - x_0)(a_1 + (x - x_1)(a_2 + (x - x_2)(a_3 + \cdots + a_n(x - x_{n-1})))) \end{aligned}$$

Given the data x_i and a_j for $i = 0, 1, \cdots, n$ the following pseudo-code evaluates the Newton's polynomial $p = P_n(x)$ effectively in $O(n)$.

- Initialize $p := a_n$
- for $k = n-1, n-2, \cdots, 0$
  - $p := p(x - x_k) + a_k$

In-class exercise:

Consider the following dataset:

x_i	-1	0	1
y_i	0	1	1

Derive the interpolation polynomial $P(x)$ by Newton's divided difference.

Homework:

1. The first homework will be released this Friday and is due next Sunday at 11:59 pm.

2. Late homework usually will not be accepted without valid justification (e.g., doctor's notes). I understand that you are busy and might forget to do your homework. If it is your first late homework, it will be accepted, but a one-point penalty will be applied for every hour it is late.

3. Homework may contain a written and a coding part. For the written part, you are welcome to use LaTeX or just handwriting; however, your writing must be readable.

4. Coding must be well commented.

5. Unless otherwise specified, a report is required for the coding part, in which you should simply attach a snapshot of your outputs.

6. Collaboration is encouraged; you should write down your collaborators' names when you submit your homework. Everyone should write their own work.

7. The use of AI is allowed; however, you MUST indicate in your submission that you used AI-assisted tools.