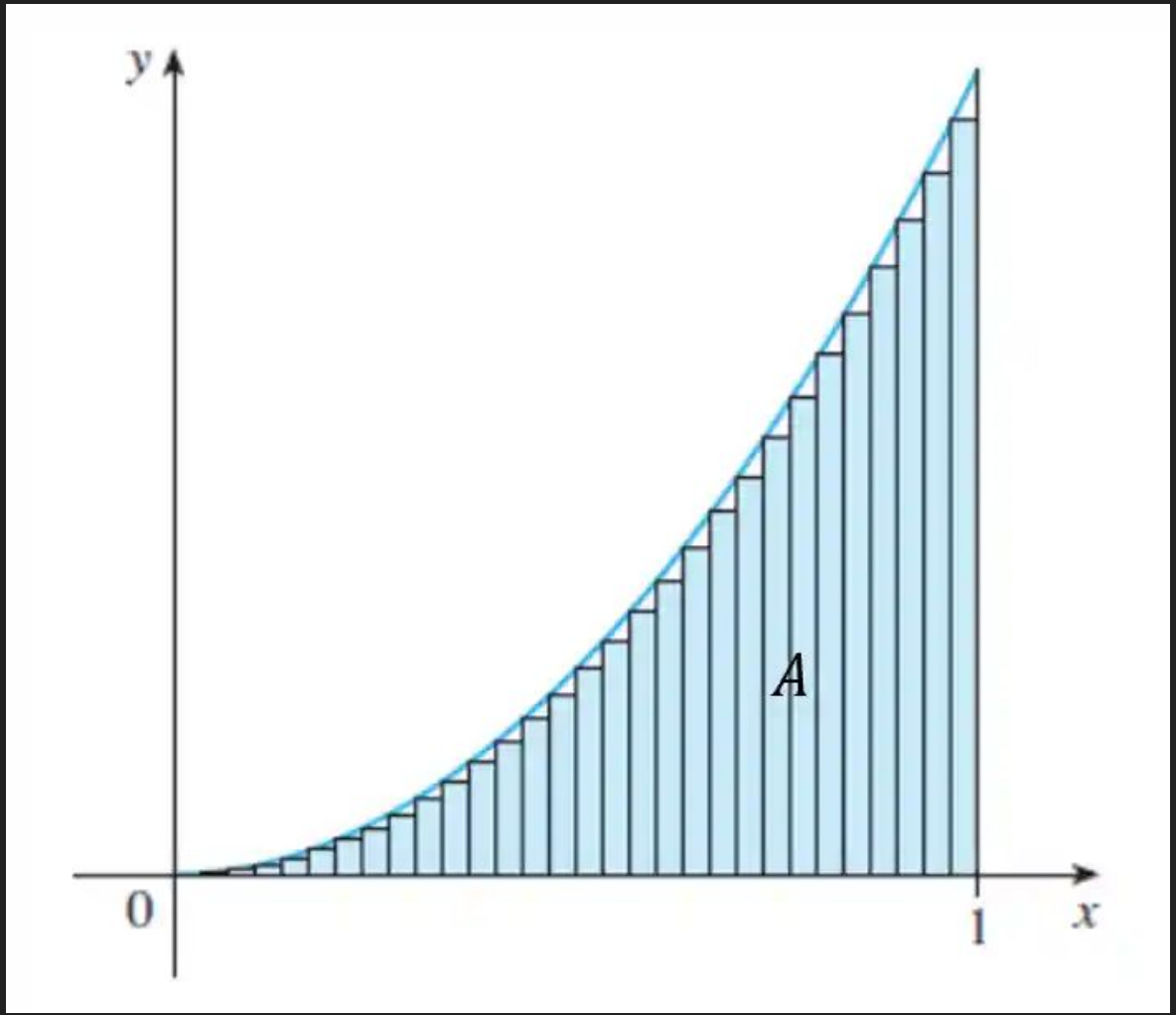
Implementação do conjunto de técnicas estudadas na disciplina para Integração Numérica, interpolação e ajuste de curvas

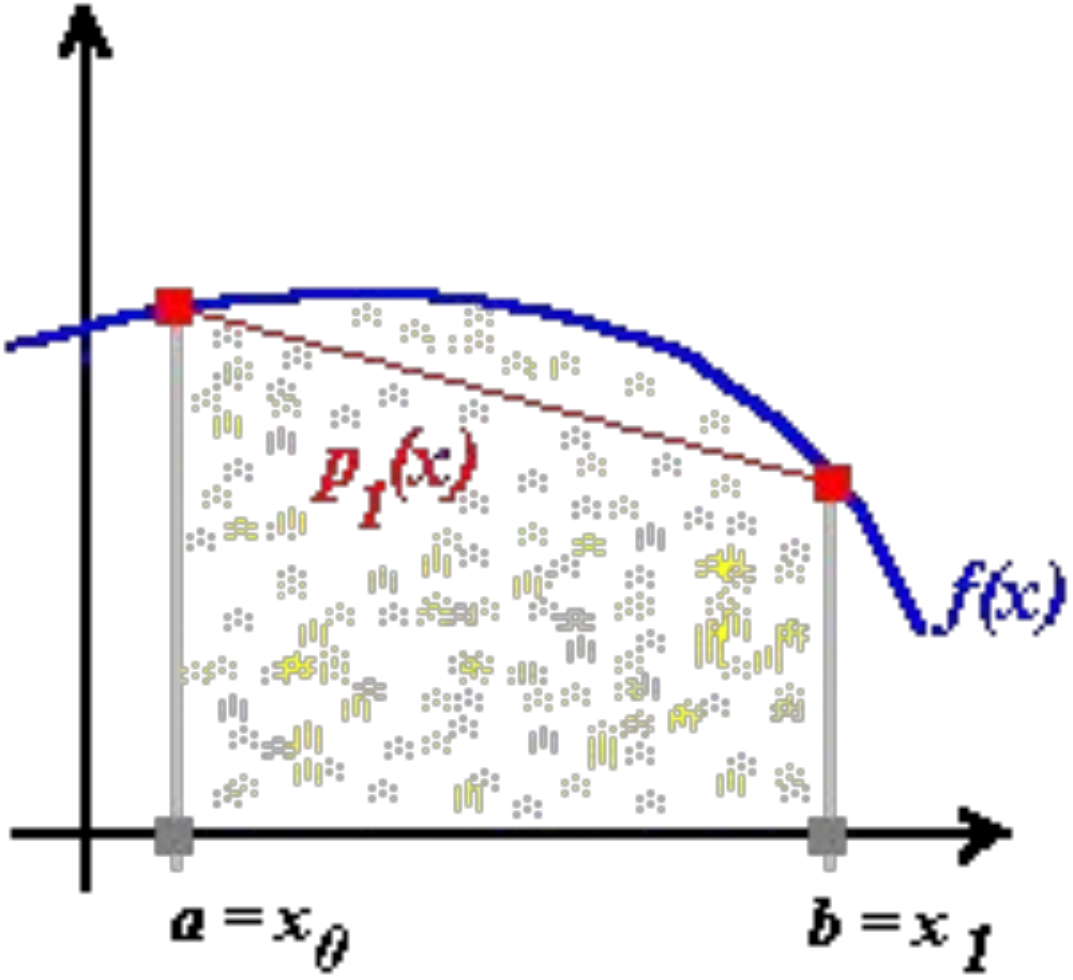
Heitor Freitas Ferreira

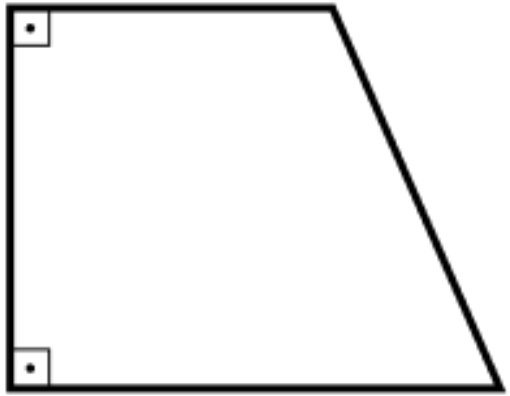
Integração numérica

Objetivo da técnica

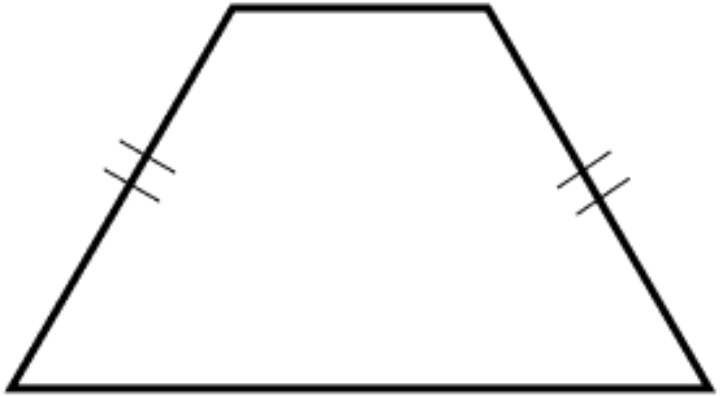
Dada uma função, calcular a área da mesma em um intervalo


y
A
0
1
x

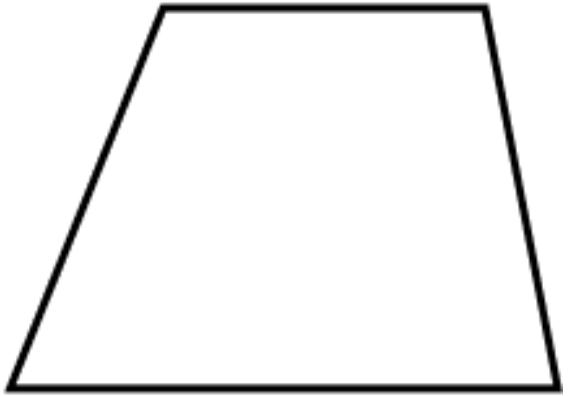

$p_1(x)$
$f(x)$
$a = x_0$
$b = x_1$



Trapézio Retângulo

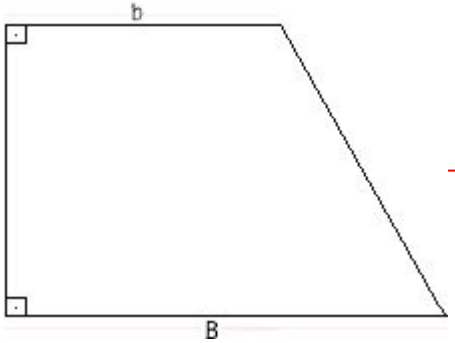


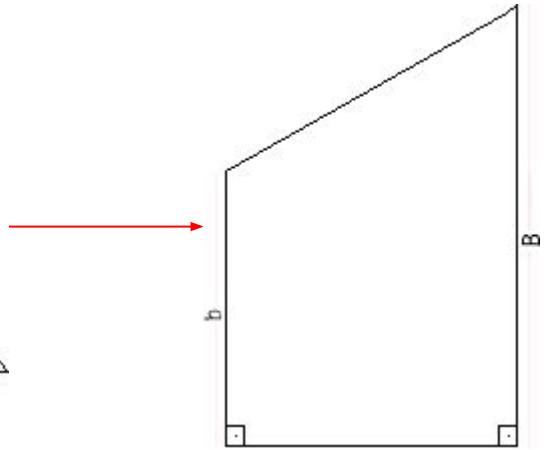
Trapézio Isósceles



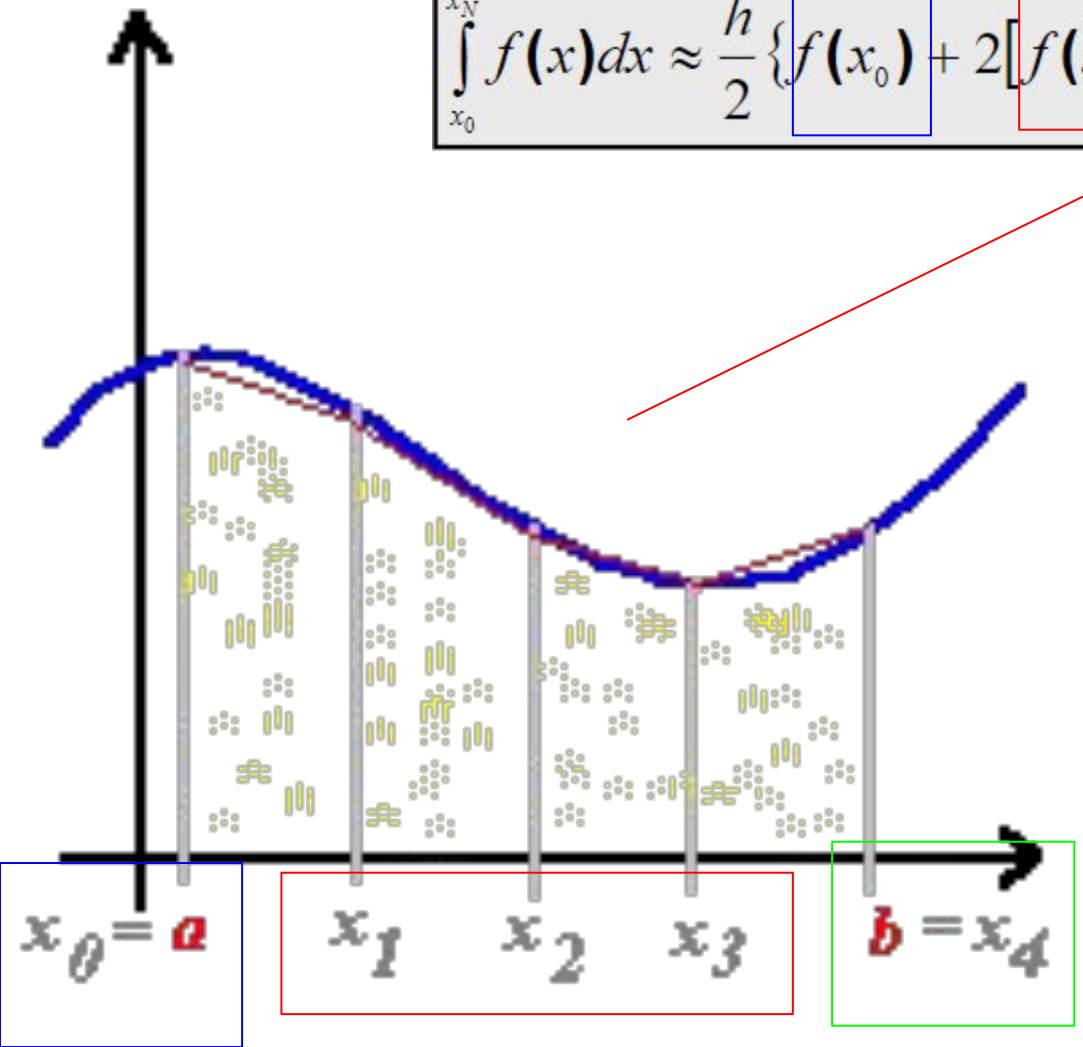
Trapézio Escaleno

$$A = \frac{(B+b).h}{2}$$





$$\int_{x_0}^{x_N} f(x)dx \approx \frac{h}{2}\left\{f(x_0) + 2\left[f(x_1) + f(x_2) + \ldots + f(x_{N-1})\right] + f(x_N)\right\}$$



Algoritmo

```
def aproximar_integral(
    funcao: callable,
    intervalo: tuple(float, float),
    num_trapezios: int = 1000
):
    inicio_intervalo = min(intervalo)
    fim_intervalo = max(intervalo)
    h = (fim_intervalo - inicio_intervalo) / num_trapezios
    soma = 0
    while inicio_intervalo < fim_intervalo:
        soma += funcao(inicio_intervalo)
        inicio_intervalo += h
    return (h / 2) * (funcao(inicio_intervalo) +
                      2 * soma + funcao(fim_intervalo))
```

Como obter a função

Função lambda

```
def ler_lambda() -> callable:
    while True:
        try:
            return eval(f"lambda x:{input('Insira a f(x): ')}")
        except:
            print("Função inválida!")
```

main

```
def main():
    f = ler_lambda()
    a = float(input("Digite INICIO o do intervalo: "))
    b = float(input("Digite FIM o do intervalo: "))

    print("Integral aproximada:", aproximar_integral(
        funcao=f,
        intervalo=(a, b)
    ))
```

Interpolação Polinomial

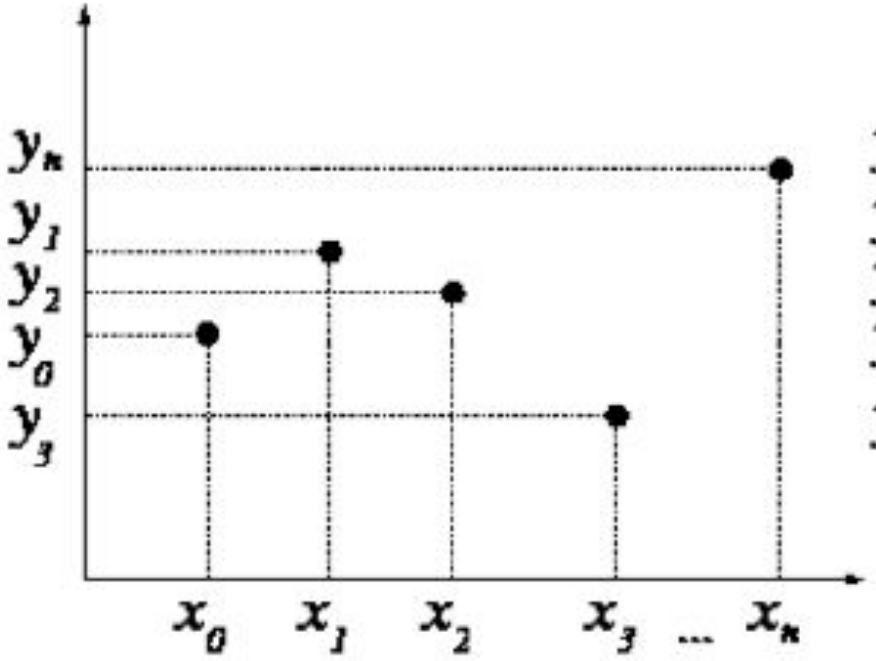
Objetivo da técnica

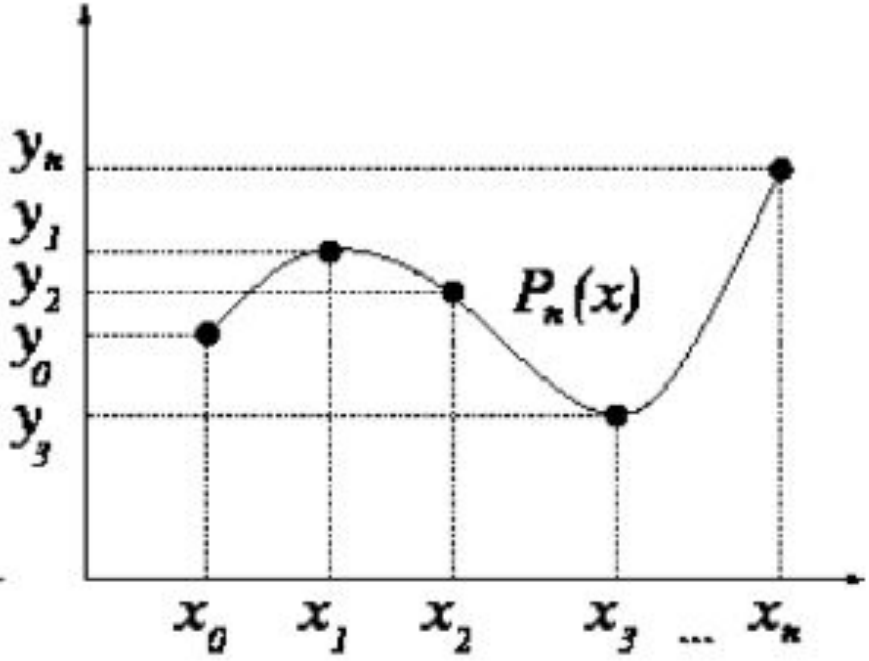
Dada uma lista de x, e uma lista com os respectivos f(x)

Obter um valor de um x arbitrário e desconhecido

Usado quando os dados são pouco ruidosos

x_i	0	1,5	3,0	4,5	6,0
$f(x_i)$	0,001	0,016	0,028	0,046	0,057





Forma de Lagrange

- Dado pela função genérica p(x)
- p(x) dá o valor que do que seria o a f(x) que é desconhecida
- L(i,x) é um produtório dado pela função abaixo

$$\mathbf{p(x) = \sum_{i=0}^{n} L_i(x).f(x_i)}$$

$$L_i(x) = \frac{(x - x_0) \cdot (x - x_1) \ldots (x - x_{i-1}) \cdot (x - x_{i+1}) \ldots (x - x_n)}{(x_i - x_0) \cdot (x_i - x_1) \ldots (x_i - x_{i-1}) \cdot (x_k - x_{i+1}) \ldots (x_i - x_n)}$$

Função do somatório

```
def interpolar_funcao(
x_valores: list[float],y_valores: list[float],x_interpolar: float
) -> float:
    grau = len(x_valores)
    def L(i: int, x: float) -> float:
        pass # próximo slide
    return sum(
        y_valores[i] * L(i, x_interpolar)
        for i in range(grau)
    )
```

$$p(x) = \sum_{i=0}^{n} L_i(x).f(x_i)$$

Função do produtório interno

```
def L(i: int, x: float) -> float:
    return produtorio(
        (x - x_valores[j])
        /
        (x_valores[i] - x_valores[j])
        for j in range(grau)
        if j != i  # Divizão por zero!
    )
```

$$L_i(x) = \frac{(x-x_0)\cdot(x-x_1)\ldots(x-x_{i-1})\cdot(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)\cdot(x_i-x_1)\ldots(x_i-x_{i-1})\cdot(x_k-x_{i+1})\ldots(x_i-x_n)}$$

Leitura dos dados

```
def newList(message: str = "") -> list[float]:
    print(message, end="")
    lista = []
    while True:
        try:
            valor =float(input("Digite um número (ou 'q' para sair): "))
            lista.append(valor)
        except ValueError:
            break
    return lista
```

main

```
def main():
    x_valores = newList("Digite os valores de x")
    y_valores = newList("Digite os valores de x")
    # Valor a ser interpolado
    x_interpolar = float(input("Digite o valor a ser interpolado:"))
    print("Valor interpolado:", interpolar_funcao(x_valores, y_valores,
x_interpolar))
```

Ajuste de curva

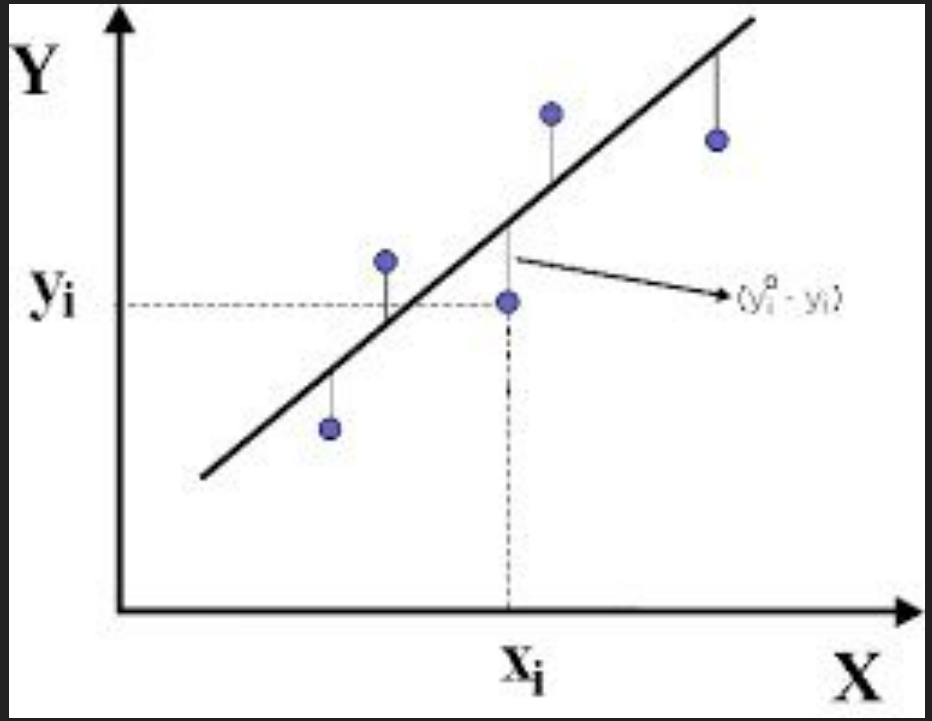
Objetivo da técnica

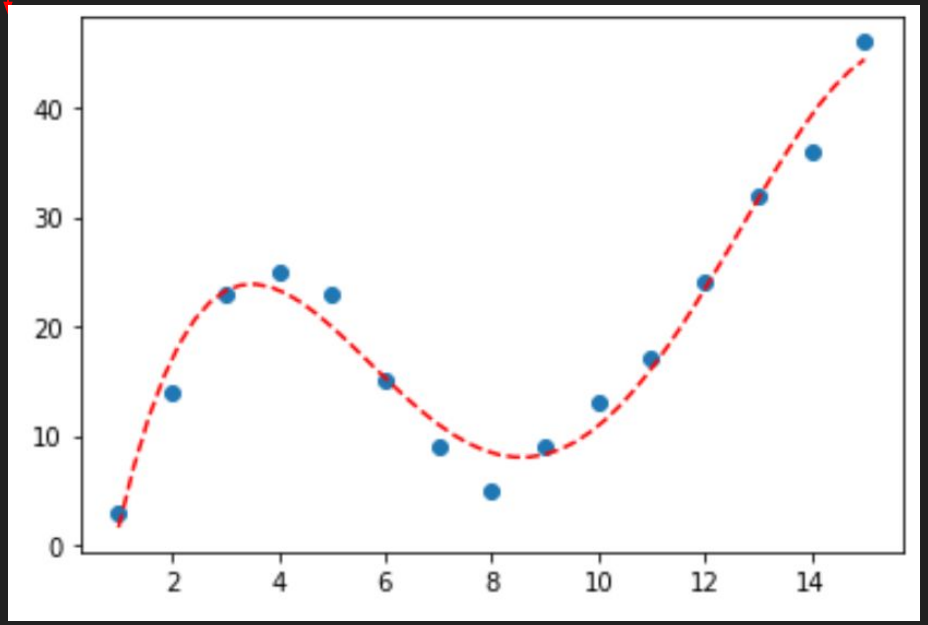
Dada uma lista de x, uma lista com os respectivos f(x) e um grau escolhido de polinômio

Obter uma função polinomial que diminua o erro para todas os pontos

Usado quando os dados são ruidosos

x_i	0	1,5	3,0	4,5	6,0
$f(x_i)$	0,001	0,016	0,028	0,046	0,057





$$
\begin{cases}
n a_0 + \left(\sum_{i=1}^{n} x_i\right) a_1 + \left(\sum_{i=1}^{n} x_i^2\right) a_2 = \sum_{i=1}^{n} y_i \\
\left(\sum_{i=1}^{n} x_i\right) a_0 + \left(\sum_{i=1}^{n} x_i^2\right) a_1 + \left(\sum_{i=1}^{n} x_i^3\right) a_2 = \sum_{i=1}^{n} x_i y_i \\
\left(\sum_{i=1}^{n} x_i^2\right) a_0 + \left(\sum_{i=1}^{n} x_i^3\right) a_1 + \left(\sum_{i=1}^{n} x_i^4\right) a_2 = \sum_{i=1}^{n} x_i^2 y_i
\end{cases}
$$

São conhecido os valores de x e y, logo seus diferentes somatórios também

Os coeficientes a serão os coeficientes da função obtida como resultado

Calcular a matriz dos somatórios e dos resultados

```
def matriz coefientes (x_valores, y_valores, dimensao):
    m = len(x valores)
    A = np.empty((dimensao, dimensao))
    b = np.empty((dimensao))
    # Somatórios de x^i, i = 0, 1, ..., dimensao+1
    somatorios = [sum([x valores[k] **i for k in range(m)]) for i in range(dimensao+2)]
    # Preenche a matriz de dispersão
    for i in range(dimensao):
        for j in range(i, dimensao):
            A[i,j] = somatorios[i+j]
            if i != j:
                A[j,i] = A[i,j]
    # Preenche o vetor de resultados da matriz de dispersão * coeficientes
    # Cada posição do vetor é um somatório de x^i * y
    for i in range(dimensao):
        b[i] = sum([y_valores[k] *x_valores[k] **i for k in range(m)])
    return A, b
```

Achar os coeficientes resolvendo por Gauss

```
def gauss(A, b):
        n = len(A)
        # Etapa de eliminação
        for i in range(n):
            pivo = A[i][i]
            # Dividir a linha i pela diagonal
principal (pivô)
            for j in range(i, n):
                A[i][j] /= pivo
            b[i] /= pivo
            # Zerar elementos abaixo do pivô
            for k in range(i + 1, n):
                fator = A[k][i]
                for j in range(i, n):
                    A[k][j] -= fator *
A[i][j]
                b[k] -= fator * b[i]
```

```
# Etapa de retrosubstituição
    coeficientes = [0] * n
    for i in range(n - 1, -1, -1):
        coeficientes[i] = b[i]
        for j in range(i + 1, n):
            coeficientes[i] -=
A[i][j] * coeficientes[j]
    return coeficientes[::-1]
```

Leitura dos dados

```
def newList(message: str = "") -> np.array:
    print(message, end="")
    lista = []
    while True:
        try:
            valor = float(input("Digite um número (ou 'q' para sair): "))
            lista.append(valor)
        except ValueError:
            break
    return np.array(lista)
```

Função de plotagem gráfica com matplotlib.pyplot

```
def plotar(x_valores, y_valores, coeficientes):
    x_plot = np.linspace(min(x_valores), max(x_valores), 100)
    y_plot = np.polyval(coeficientes, x_plot)
    plt.scatter(x_valores, y_valores, color='red', label='Pontos
Conhecidos')
    plt.plot(x_plot, y_plot, label='Curva Ajustada')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.grid(True)
    plt.show()
```

main

```
def ajustar_curva(x_valores, y_valores, grau):
    A, b = matriz_coefientes(x_valores, y_valores, grau+1)
    return gauss(A, b)
def main():
    x_valores = newList()
    y_valores = newList()
    grau = int(input("Digite o grau do polinomio a ser ajustado"))
    coeficientes = ajustar_curva(x_valores, y_valores, grau)
    print("Coeficientes ajustados:", coeficientes)
    if input("Deseja plotar o gráfico? (s/n) ").lower() == 's':
        plotar(x_valores, y_valores, coeficientes)
```