UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI - UFSJ

DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEPEL

COORDENAÇÃO DE ENGENHARIA ELÉTRICA - COELE

GABRIEL AUGUSTO SILVA BATISTA

TRABALHO 7

São João del-Rei – MG

Dezembro de 2023

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

def gauss_elimination(A, b):
    n = len(A)
    M = A.tolist()  # Convertendo o array numpy para uma lista

    i = 0
    for x in M:
        x.append(b[i])
        i += 1

    for k in range(n):
        for i in range(k,n):
            if abs(M[i][k]) > abs(M[k][k]):
                M[k], M[i] = M[i],M[k]
            else:
                pass

        for j in range(k+1,n):
            q = float(M[j][k]) / M[k][k]
            for m in range(k, n+1):
                M[j][m] -=  q * M[k][m]

    return M

def back_substitution(A):
    n = len(A)
    x = [0 for i in range(n)]

    x[n-1] = A[n-1][n] / A[n-1][n-1]
    for i in range(n-2, -1, -1):
        x[i] = A[i][n]

        for j in range(i+1, n):
            x[i] = x[i] - A[i][j]*x[j]

        x[i] /= A[i][i]

    return x
```

Aqui é importado as bibliotecas necessárias: numpy para operações matemáticas e matplotlib.pyplot para a plotagem dos gráficos.

Em seguida, definimos várias funções necessárias para a aplicação do método:

gauss_elimination(A, b): Esta função realiza a eliminação de Gauss em uma matriz ampliada formada pela matriz A e o vetor b. A matriz resultante é retornada.

back_substitution(A): Esta função realiza a substituição retroativa em uma matriz triangular superior A para resolver um sistema de equações lineares. O vetor de soluções é retornado.

```python
def coef_newton(X, Y):
    n = len(X)
    coef = np.zeros([n, n])
    coef[:,0] = Y

    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = (coef[i+1][j-1] - coef[i][j-1]) / (X[i+j] - X[i])

    return coef[0, :]

# Função para calcular o valor do polinômio interpolador de Newton em um ponto
def eval_newton(coef, X, x):
    n = len(X) - 1
    p = coef[n]
    for k in range(1,n+1):
        p = coef[n-k] + (x - X[n-k])*p
    return p
```

coef_newton(X, Y): Esta função calcula os coeficientes do polinômio interpolador de Newton para um conjunto de pontos (X, Y). Os coeficientes são retornados.

eval_newton(coef, X, x): Esta função avalia o polinômio interpolador de Newton em um ponto x usando os coeficientes coef e os pontos X.

```python
x = [0.000, 2.000, 4.000, 6.000, 8.000, 10.000, 12.000, 14.000, 16.000]
y = [1.000, 7.000, 21.000, 22.000, 34.000, 34.500, 35.000, 64.500, 65.000]
x2,x3,x4,xy,x2y = [], [], [], [], []

for i in range(len(x)):
    x2.append(x[i]**2)
    x3.append(x[i]**3)
    x4.append(x[i]**4)
    xy.append(x[i]*y[i])
    x2y.append(x2[i]*y[i])

sumx = sum(x)
sumy = sum(y)
sumx2 = sum(x2)
sumx3 = sum(x3)
sumx4 = sum(x4)
sumxy = sum(xy)
sumx2y = sum(x2y)

A = np.array([[9, sumx, sumx2], [sumx, sumx2, sumx3], [sumx2, sumx3, sumx4]], dtype=float)
b = np.array([sumy, sumxy, sumx2y], dtype=float)

Mampliada = gauss_elimination(A, b)
coef = back_substitution(Mampliada)

f = np.poly1d(coef[::-1])  # Invertendo a ordem dos coeficientes
print("Ajustada: ")
print("Os coeficientes do polinômio são:", coef[::-1])
print(f)
print(f'\n')

```

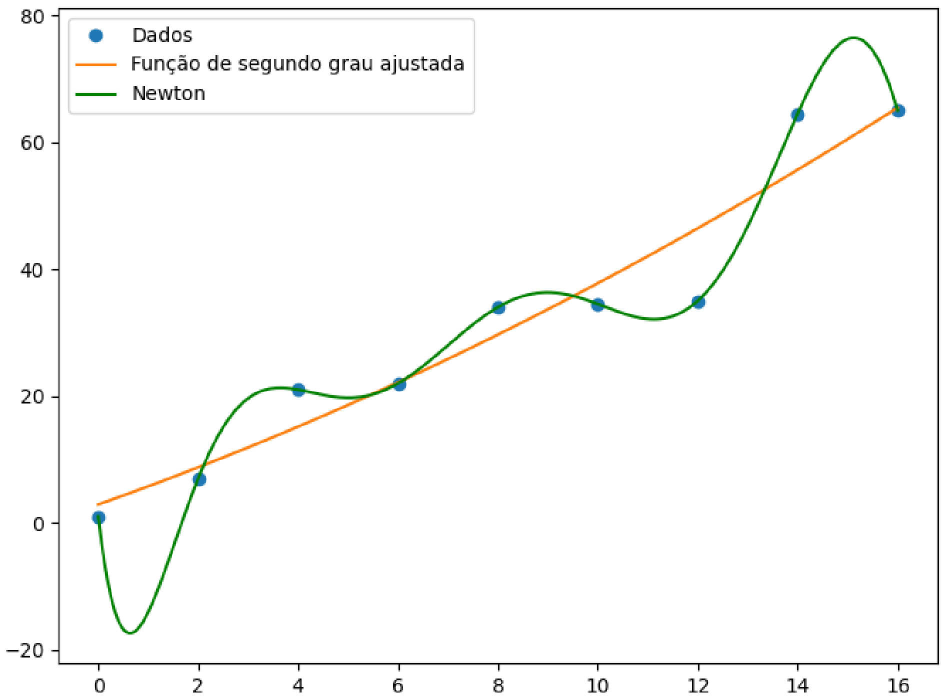
Depois de definir as funções, definimos os pontos de dados x e y. Em seguida, calculamos os somatórios necessários para o ajuste da função de segundo grau, como x2, x3, x4, xy e x2y.

```python
# Polinômio interpolador de Newton
coef_Newton = coef_newton(X,Y)
fc = np.poly1d(coef_Newton)
print("Newton: ")
print("Os coeficientes do interpolador são:",coef_Newton)
print(fc)

X_plot = np.linspace(np.min(X),np.max(X),500)
y_new = f(X_plot)

# Plot dos dados e da função ajustada
plt.figure(figsize=(8, 6))
plt.plot(x, y, 'o', label='Dados')
plt.plot(X_plot, y_new, '-', label='Função de segundo grau ajustada')
Y_plot_Newton = [eval_newton(coef_Newton,X,x) for x in X_plot]
plt.plot(X_plot,Y_plot_Newton,'g-', label='Newton') # função polinomial de Newton
plt.plot()
plt.legend()
plt.show()
```

Aqui é resolvido o sistema de equações lineares para os coeficientes da função de segundo grau usando a eliminação de Gauss e a substituição retroativa.

Por fim, calculamos os coeficientes do polinômio interpolador de Newton e avalia ambos os polinômios em um conjunto de pontos para plotagem. O gráfico resultante mostra os pontos de dados originais, a função de segundo grau ajustada e o polinômio interpolador de Newton.

```
Ajustada:
Os coeficientes do polinômio são: [0.0703463203463204, 2.782792207792207, 2.915151515151514]
         2
0.07035 x + 2.783 x + 2.915


Newton:
Os coeficientes do interpolador são: [ 1.00000000e+00  3.00000000e+00  1.00000000e+00 -4.37500000e-01
  1.17187500e-01 -2.38281250e-02  3.73263889e-03 -4.16976687e-04
  2.69329737e-05]
   8     7     6          5          4           3            2
1 x + 3 x + 1 x - 0.4375 x + 0.1172 x - 0.02383 x + 0.003733 x - 0.000417 x + 2.693e-05
```



Ambos os métodos forneceram resultados satisfatórios. No entanto, a função de segundo grau ajustada e o polinômio interpolador de Newton diferem ligeiramente. Isso é esperado, pois a interpolação de Newton gera um polinômio que passa exatamente por todos os pontos de dados, enquanto o ajuste de função de segundo grau busca minimizar o erro quadrático total.