UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI - UFSJ
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEPEL
COORDENAÇÃO DE ENGENHARIA ELÉTRICA - COELE

GABRIEL AUGUSTO SILVA BATISTA

TRABALHO 8

São João del-Rei – MG
Dezembro de 2023

O código começa importando duas bibliotecas: numpy e math. A biblioteca numpy é usada para operações numéricas eficientes, enquanto a biblioteca math fornece funções matemáticas.

```
import numpy as np
import math
```

Em seguida, o código define duas funções: f(t, y) e y_exacta(t). A função f(t, y) representa a equação diferencial ordinária (EDO) que queremos resolver. A função y_exacta(t) representa a solução exata da EDO, que é usada para calcular o erro dos métodos numéricos.

```
# Função da EDO
def f(t, y):
    return y + math.exp(2*t) + math.sin(t) + math.cos(t)

# Função exata
def y_exacta(t):
    return math.exp(2*t) - math.cos(t)

```

O código então implementa três métodos numéricos para resolver a EDO: o método de Euler, o método de Euler melhorado e o método de Runge-Kutta de 4ª ordem. Cada método é implementado como uma função que toma um passo h, um tempo inicial t0, um valor inicial y0 e um tempo final t, e retorna a aproximação da solução no tempo t.

```
# Método de Euler
def euler(h, t0, y0, t):
    while t0 < t:
        y0 = y0 + h*f(t0, y0)
        t0 = t0 + h
    return y0

# Método de Euler melhorado
def euler_melhorado(h, t0, y0, t):
    while t0 < t:
        k1 = h*f(t0, y0)
        k2 = h*f(t0 + h, y0 + k1)
        y0 = y0 + 0.5*(k1 + k2)
        t0 = t0 + h
    return y0

# Método de Runge-Kutta de 4ª ordem
def runge_kutta(h, t0, y0, t):
    while t0 < t:
        k1 = h*f(t0, y0)
        k2 = h*f(t0 + 0.5*h, y0 + 0.5*k1)
        k3 = h*f(t0 + 0.5*h, y0 + 0.5*k2)
        k4 = h*f(t0 + h, y0 + k3)
        y0 = y0 + (k1 + 2*k2 + 2*k3 + k4)/6
        t0 = t0 + h
    return y0

```

O código então define alguns parâmetros iniciais e calcula os passos para cada método com base no intervalo de tempo desejado. Ele usa esses passos para calcular as aproximações da solução usando cada método, bem como o valor exato no tempo t.

```python
# Parâmetros iniciais
t0 = 0
y0 = 0
t = 2

# Passos para cada método
h_euler = (t - t0) / 16
h_euler_melhorado = (t - t0) / 8
h_runge_kutta = (t - t0) / 4

# Calculando os valores
y_euler = euler(h_euler, t0, y0, t)
y_euler_melhorado = euler_melhorado(h_euler_melhorado, t0, y0, t)
y_runge_kutta = runge_kutta(h_runge_kutta, t0, y0, t)
y_exato = y_exacta(t)

```

Finalmente, o código calcula os erros de cada método subtraindo a aproximação da solução do valor exato, e imprime os resultados.

```python
# Calculando os erros
e_euler = y_exato - y_euler
e_euler_melhorado = y_exato - y_euler_melhorado
e_runge_kutta = y_exato - y_runge_kutta

# Imprimindo os resultados
print('Método de Euler:', y_euler, ', Erro:', e_euler)
print("Método de Euler Melhorado: ", y_euler_melhorado, ', Erro:', e_euler_melhorado)
print("Método de Runge-Kutta: ", y_runge_kutta, ', Erro:', e_runge_kutta)
print("Valor exato: ", y_exato)
```

```
Método de Euler: 44.5030590083383 , Erro: 10.51123786135308
Método de Euler Melhorado:  54.276622272610545 , Erro: 0.7376745970808329
Método de Runge-Kutta:  54.96066266439705 , Erro: 0.05363420529432972
Valor exato:  55.01429686969138
```

Os métodos de Euler e Euler melhorado são métodos de primeira e segunda ordem, respectivamente, o que significa que o erro é proporcional a h2 e h3, respectivamente. O método de Runge-Kutta de 4ª ordem é um método de quarta ordem, o que significa que o erro é proporcional a h5. Portanto, espera-se que o método de Runge-Kutta de 4ª ordem seja o mais preciso para um dado tamanho de passo h.