Programa: EDO (ecuaciones diferenciales ordinarias)

por: Ovalle D., Bernal M., Posada J

modificado por: Santana R.

A. Introducción

El código proporcionado es una implementación de métodos numéricos para resolver ecuaciones diferenciales ordinarias (EDOs). Los métodos implementados son el método de Euler y el método de Runge-Kutta de cuarto orden (RK4). También se incluyen versiones de estos métodos para sistemas de EDOs.

Los métodos de euler y RK4 de la clase EDO fueron copiados del libro **Métodos Numéricos con Python**, *Ovalle D., Bernal M., Posada J.*, Editorial Politecnico Grancolombiano, (2021). En cuanto a los métodos sis_euler y sis_RK4 fueron modificados, de forma que se adapten al problema.

B. EDO

Entrada:

- a: Valor inicial del intervalo de integración.
- b: Valor final del intervalo de integración.
- y0: Valor inicial de la función o vector de funciones.
- N: Número de pasos de integración.
- f: Función que define la ecuación diferencial o el sistema de ecuaciones diferenciales

Salida:

- euler(): w. Devuelve el valor final de las variables dependientes.
- RK4(): w. Devuelve el valor final de las variables dependientes.
- sis_euler(): Devuelve un array que contiene el tiempo y los valores finales de las variables dependientes.
- sis_RK4(): Devuelve un array que contiene el tiempo y los valores finales de las variables dependientes

C. Programa

Método de Euler

El método de Euler es un método numérico para resolver ecuaciones diferenciales de primer orden. En este caso, el método de Euler se implementa en el método euler() de la clase EDO. El método calcula la solución aproximada de la ecuación diferencial utilizando el paso de integración h y devuelve el valor final de la solución. Durante el proceso de cálculo, se imprimen los valores de t y w en cada paso de integración.

Método de Runge-Kutta de cuarto orden (RK4)

El método de Runge-Kutta de cuarto orden (RK4) es un método numérico más preciso para resolver ecuaciones diferenciales de primer orden. En este caso, el método RK4 se implementa en el método RK4() de la clase EDO. El método calcula la solución aproximada de la ecuación diferencial utilizando el paso de integración h y devuelve el valor final de la solución. Durante el proceso de cálculo, se imprimen los valores de t y w en cada paso de integración.

Métodos para sistemas de ecuaciones diferenciales

Además de los métodos anteriores, la clase EDO también implementa versiones de los métodos de Euler y RK4 para sistemas de ecuaciones diferenciales. Estos métodos se llaman sis_euler() y sis_RK4() respectivamente. Estos métodos calculan la solución aproximada de un sistema de ecuaciones diferenciales utilizando el paso de integración h y devuelven el valor final de la solución en forma de un vector.

En cada uno de los métodos, se imprimen los valores de t y las componentes de la solución en cada paso de integración.

D. Código

```
import math as mt
import numpy as np

class EDO():

    def __init__(self, a, b, y0, N, f):
        self.a = a
        self.b = b
        self.y0 = y0
        self.N = N
        self.f = f


    def euler(self):
```

```
    h = (self.b - self.a)/self.N
    t = self.a
    w = self.y0
    print("t0 = {0:.2f}, w0 = {1:.12f}".format(t, w))
    for i in range(1, self.N+1):
        w = w + h*self.f(t, w)
        t = self.a + i*h
        print("t{0:<2} = {1:.2f}, w{0:<2} = {2:.12f}".format(i, t, w))
    return w

def RK4(self):
    h = (self.b - self.a)/self.N
    t = self.a
    w = self.y0
    print("t0 = {0:.2f}, w0 = {1:.12f}".format(t, w))

    for i in range(1, self.N+1):
        k1 = h*self.f(t, w)
        k2 = h*self.f(t + h/2, w + k1/2)
        k3 = h*self.f(t + h/2, w + k2/2)
        k4 = h*self.f(t + h, w + k3)
        w = w + (k1 + 2*k2 + 2*k3 + k4)/6
        t = self.a + i*h
        print("t{0:<2} = {1:.2f}, w{0:<2} = {2:.12f}".format(i, t, w))
    return w

def sis_euler(self):
    h = (self.b - self.a)/self.N
    t = self.a
    w = np.array(self.y0)
    v = np.concatenate([[t],w])
    print("t_0 = {0:.2f}, u1_0 = {1:.12f}, u2_0 = {2:.12f}, u3_0 = {3:.12f}".format(t,

    for i in range(1, self.N+1):
        w = w + h*self.f(t, w)
        t = self.a + i*h
        print("t{0:<2} = {1:.2f}, u1_{0:<2} = {2:.12f}, u2_{0:<2} = {3:.12f}, u3_{0:<2
    return np.concatenate([[t],w])

def sis_RK4(self):
    h = (self.b - self.a)/self.N
    t = self.a
    w = np.array(self.y0)
    v = np.concatenate([[t],w])
    print("t_0 = {0:.2f}, u1_0 = {1:.12f}, u2_0 = {2:.12f}, u3_0 = {3:.12f}".format(t,

    for i in range(1, self.N+1):
        k1 = h*self.f(t, w)
        k2 = h*self.f(t + h/2, w + k1/2)
        k3 = h*self.f(t + h/2, w + k2/2)
        k4 = h*self.f(t + h, w + k3)
```

```
w = w + (k1 + 2*k2 + 2*k3 + k4)/6
t = self.a + i*h
```