O método Runge-Kutta para sistemas 2×2

Doherty Andrade

www.metodosnumericos.com.br

1 Método Runge-Kutta de ordem 4 para sistemas 2 por 2

O método de Runge-Kutta de ordem 4 pode ser estendido para sistemas de equações.

Consideremos o caso de um sistema de duas equações diferenciais.

Suponha que temos um sistema de EDO's dado por

$$\begin{cases} y' = f(x, y, z) \\ z' = g(x, y, z), \\ y(x_0) = y_0, \\ z(x_0) = z_0. \end{cases}$$

Para usar o método de RK4 devemos calcular os coeficientes

$$\begin{aligned} K_1 &= hf(x_k, y_k, z_k), \\ L_1 &= hg(x_k, y_k, z_k), \\ K_2 &= hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1, z_k + \frac{1}{2}L_1), \\ L_2 &= hg(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1, z_k + \frac{1}{2}L_1), \\ K_3 &= hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2, z_k + \frac{1}{2}L_2), \\ L_3 &= hg(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2, z_k + \frac{1}{2}L_2), \\ K_4 &= hf(x_k + h, y_k + K_3, z_k + L_3), \\ L_4 &= hg(x_k + h, y_k + K_3, z_k + L_3), \end{aligned}$$

calculados nesta ordem. Em seguida calculamos y_{k+1} e z_{k+1} usando

$$y_{k+1} = y_k + \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right) \tag{1}$$

e

$$z_{k+1} = z_k + \frac{1}{6}\left(L_1 + 2L_2 + 2L_3 + L_4\right), \tag{2}$$

para obter as soluções numéricas, exatamente como no caso de EDO de ordem 1.

Para usar o procedimento abaixo basta digitar RungeKuta22(f1,f2,x0,y0,z0,xf,n), antes entrar com os dados.

```
In [15]: import numpy as np
         import matplotlib.pyplot as plt

         def RungeKuta22(f1,f2,x0,y0,z0,xf,n):

             h = (xf-x0)/(n-1)
             x = np.linspace(x0,xf,n)
             y = np.zeros([n,1])
             z = np.zeros([n,1])

             y[0] = y0
             z[0] = z0

             for i in range(1,n):
                 k1 = h*f1(x[i-1],y[i-1],z[i-1])
                 l1 = h*f2(x[i-1],y[i-1],z[i-1])
                 k2 = h*f1(x[i-1]+h/2,y[i-1]+k1/2,z[i-1]+l1/2)
                 l2 = h*f2(x[i-1]+h/2,y[i-1]+k1/2,z[i-1]+l1/2)
                 k3 = h*f1(x[i-1]+h/2,y[i-1]+k2/2,z[i-1]+l2/2)
                 l3 = h*f2(x[i-1]+h/2,y[i-1]+k2/2,z[i-1]+l2/2)
                 k4 = h*f1(x[i-1]+h, y[i-1]+k3,z[i-1]+l3)
                 l4 = h*f2(x[i-1]+h, y[i-1]+k3,z[i-1]+l3)

                 y[i] = y[i-1] + (k1 + 2*k2 + 2*k3 + k4)/6
                 z[i] = z[i-1] + (l1 + 2*l2 + 2*l3 + l4)/6

             plt.plot(x,y,label='aprox. Y')
             plt.plot(x,z,label='aprox. Z')
             plt.xlabel("X valores")
             plt.ylabel("Y,Z valores")
             plt.legend()
             plt.title("Runge-Kutta para sistemas 2 por 2")
             plt.show()
```

Exemplo 1:

$$\begin{cases} y' = z \\ z' = 0.05z - 0.15y \\ y(0) = 1, \\ z(0) = 0. \end{cases}$$

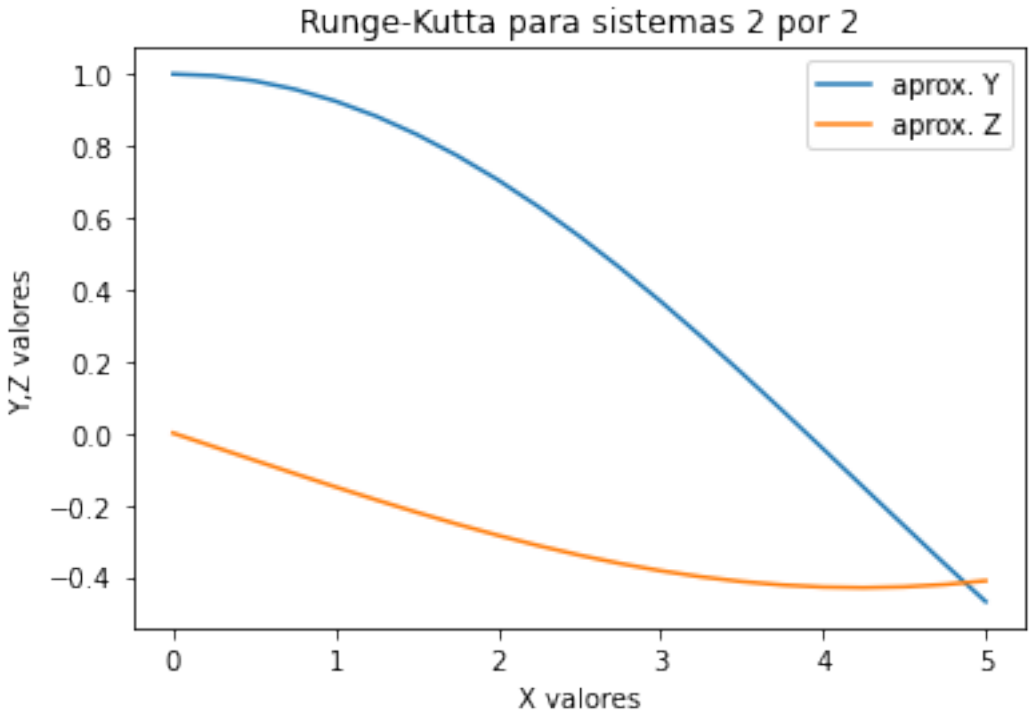
Utilizando o método de Runge-Kutta para sistemas, obtemos os seguintes dados para o intervalo $[0,5]$. Neste exemplo temos $f_1(x,y,z) = z$ e $f_2(x,y,z) = 0.05z - 0.15y$.

```
In [16]: def f1(x,y,z):
             return z

         def f2(x,y,z):
             return 0.05*z-0.15*y
```

```
x0 = 0.0
y0 = 1.0
z0 = 0.0
xf = 5.0
n  = 21  # -> 20 subintervalor

RungeKuta22(f1,f2,x0,y0,z0,xf,n,)
```



Exemplo 2: Considere o PVI dado por

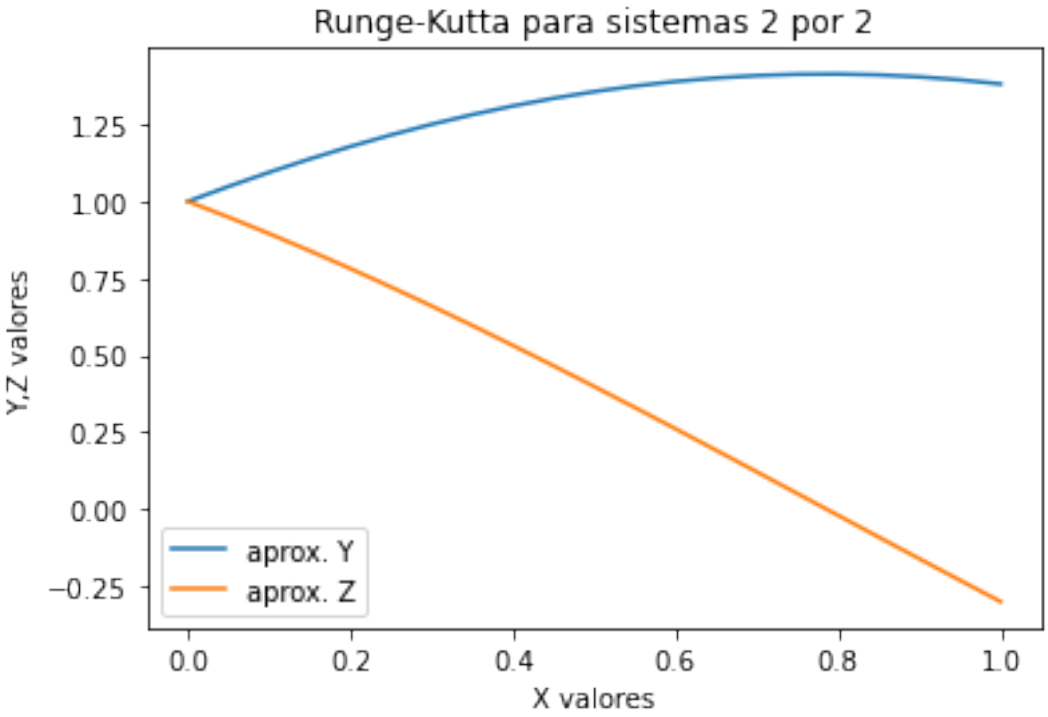
$$\begin{cases} y' = z, x \in [0,1] \\ z' = -y \\ y(0) = 1, \\ z(0) = 1. \end{cases}$$

```
In [17]: def f1(x,y,z):
             return z

         def f2(x,y,z):
             return -y
```

```
x0 = 0.0
y0 = 1.0
z0 = 1.0
xf = 1.0
n  = 21  # 20 subintervalos.

RungeKuta22(f1,f2,x0,y0,z0,xf,n,)
```



In [ ]: