16. Use the continuation method and Runge-Kutta method of order four on the following nonlinear systems:

$$\begin{aligned} 10x_1 - 2x_2^2 + x_2 - 2x_3 - 5 &= 0 \\ 8x_2^2 + 4x_3^2 - 9 &= 0 \\ 8x_2x_3 + 4 &= 0 \end{aligned}$$

Usamos $x(0) = (0, 0, 1.5)^T$

$$f_1(x_1, x_2, x_3) = 10x_1 - 2x_2^2 + x_2 - 2x_3 - 5$$

$$f_2(x_1, x_2, x_3) = 8x_2^2 + 4x_3^2 - 9$$

$$f_3(x_1, x_2, x_3) = 8x_2x_3 + 4$$

$$J(x_1, x_2, x_3) = \begin{pmatrix} 10 & -4x_2 + 1 & -2 \\ 0 & 16x_2 & 8x_3 \\ 0 & 8x_3 & 8x_2 \end{pmatrix}$$

Evaluando en $x(0)$; $f_1(x(0)) = -8$

$f_2(x(0)) = 0 \rightarrow F(x(0)) = (-8, 0, 4)$

$f_3(x(0)) = 4$

$N = 4$; $h = 1/4 = 0.25$; $W_0 = (0, 0, 1.5)$

$$K_1 = h\left(-J(W_0)\right)^{-1} F(x_0)$$

$$K_2 = h\left[-J(W_0) + \tfrac{1}{2}K_1\right]^{-1} F(x_0)]$$

$$K_3 = h\left[-J(W_0) + \tfrac{1}{2}K_2\right]^{-1} F(x_0)]$$

$$K_4 = h\left[-J(W_0) + K_3\right]^{-1} F(X_0)]$$

$$\rightarrow \quad K_1 = h\left(-J(W_0)\right)^{-1} F(X_0)] = [0.2083333, -0.08333, 0.]$$

$$K_2 = h\left[-J(W_0) + \tfrac{1}{2} K_1\right]^{-1} F(X_0)] = [0.20880989, -0.08346, -0.00463]$$

$$K_3 = h\left[-J(W_0) + \tfrac{1}{2} K_2\right]^{-1} F(X_0)] = [0.2088289, -0.083592, -0.0046]$$

$$K_4 = h\left[-J(W_0) + K_3\right]^{-1} F(X_0)] = [0.20934358, -0.08418, -0.009404]$$

$$W_1 = W_0 + \frac{1}{6}\left[K_1 + 2K_2 + 2K_3 + K_4\right]$$

$$\rightarrow W_1 = [0.20882375, -0.08359342, 1.49533415]$$

Realizando las demas iteraciones con el codigo python

RungeKutta4.py > ...

```python
import numpy as np
def F(x):
    f1=10*x[0]-2*(x[1]**2)+x[1]-(2*x[2])-5
    f2=8*(x[1]**2)+4*(x[2]**2)-9
    f3=8*x[1]*x[2]+4
    return np.array([f1,f2,f3])
def Jacobiano(x):
    return np.array([[10,-4*x[1]+1,-2],[0,16*x[1],8*x[2]],[0,8*x[2],8*x[1]]])

# def init main
if __name__ == "__main__":
    x0 = np.array([0.,0.,1.5])
    Tolerancia = 1e-4
    N = 4
    h = 1/N
    w0 = x0
    for i in range(N):
        print(f"W_{i} = {w0}")
        k1 = h*np.dot(np.linalg.inv(-Jacobiano(w0)),F(x0))
        k2 = h*np.dot(np.linalg.inv(-Jacobiano(w0+0.5*k1)),F(x0))
        k3 = h*np.dot(np.linalg.inv(-Jacobiano(w0+0.5*k2)),F(x0))
        k4 = h*np.dot(np.linalg.inv(-Jacobiano(w0+k3)),F(x0))
        w0 = w0 + (k1+2*k2+2*k3+k4)/6
        print("K_1: ",k1)
        print("K_2: ",k2)
        print("K_3: ",k3)
        print("K_4: ",k4)
        if(np.linalg.norm(k1) < Tolerancia):
            break
    print("x*= ",w0)
    #round 4 decimals with np.round
    print("f(x*)= ",np.round(F(w0),4))
```

```
PS C:\Users\Dany\Desktop\Codigos Numerico\Numerico1-main>
igos Numerico/Numerico1-main/RungeKutta4.py"
W_0 = [0.  0.  1.5]
K_1:  [ 0.20833333 -0.08333333  0.        ]
K_2:  [ 0.20880989 -0.08346213 -0.00463679]
K_3:  [ 0.20882289 -0.08359213 -0.00465839]
K_4:  [ 0.20934358 -0.08411868 -0.00940475]
W_1 = [ 0.20882375 -0.08359342  1.49533415]
K_1:  [ 0.20934364 -0.08411912 -0.009405   ]
K_2:  [ 0.20991386 -0.08506597 -0.01434129]
K_3:  [ 0.20993239 -0.08522044 -0.01444539]
K_4:  [ 0.21056637 -0.08666107 -0.01975785]
W_2 = [ 0.4187575  -0.16881892  1.48087812]
K_1:  [ 0.2105665  -0.08666186 -0.01975877]
K_2:  [ 0.21127561 -0.08866471 -0.02557474]
K_3:  [ 0.21130574 -0.0888923  -0.02581245]
K_4:  [ 0.21212068 -0.09165717 -0.03246738]
W_3 = [ 0.63006581 -0.25772443  1.4550447 ]
K_1:  [ 0.21212101 -0.0916593  -0.03247026]
K_2:  [ 0.21306828 -0.09536711 -0.04024032]
K_3:  [ 0.21312606 -0.09579138 -0.0407763 ]
K_4:  [ 0.21428452 -0.10100715 -0.05049625]
x*=  [ 0.84319817 -0.35355501  1.41421141]
f(x*)=  [ 0. -0. -0.]
```

→ se comprueba que la solución

$$x^* = [0.84319817, -0.35355501, 2.41421141]$$

es la solución aproximada.