```
from RK4 import RK4
import numpy as np
import matplotlib.pyplot as plt

#Forcing funciton, for this case there are no forces over time
def F(t, y1, y2):
    return 0

#dz/dx
def f1(t, y1, y2):
    return y2

#standard form dy/dt for different damping coefficients--------------------
def f2(t, y1, y2):
    c = 5
    m = 20
    k = 20
    sign = np.sign((-c*y2 - k*y1 + F(t, y1, y2))/m)
    return np.sqrt(np.abs((-c*y2 - k*y1 + F(t, y1, y2))/m))*sign

def f3(t, y1, y2):
    c = 40
    m = 20
    k = 20
    sign = np.sign((-c*y2 - k*y1 + F(t, y1, y2))/m)
    return np.sqrt(np.abs((-c*y2 - k*y1 + F(t, y1, y2))/m))*sign

def f4(t, y1, y2):
    c = 200
    m = 20
    k = 20
    sign = np.sign((-c*y2 - k*y1 + F(t, y1, y2))/m)
    return np.sqrt(np.abs((-c*y2 - k*y1 + F(t, y1, y2))/m))*sign

#-----------------------------------------------------------------------------

fu = [f1, f2]   #underdamped functions
fc = [f1, f3]   #critically damped funcs
fo = [f1, f4]   #overdamped funcs
t0 = 0          #intial time
y0 = [1, 0]     #intial conditions
#Solutions
tRange, y1 = RK4(0, y0, fu, 0.1, 15)
tRange, y2 = RK4(0, y0, fc, 0.1, 15)
tRange, y3 = RK4(0, y0, fo, 0.1, 15)

#Plotting
plt.figure()
plt.plot(tRange, y1[:, 0], label='underdamped')
plt.plot(tRange, y2[:, 0], label='critically damped')
plt.plot(tRange, y3[:, 0], label='overdamped')
plt.title('Mass spring damper system')
plt.xlabel('t')
plt.ylabel('y')
plt.legend()
plt.show()

#As the damping coefficient increases the motion of the spring goes from
# a wave like motion to an exponential decay motion. The critically damped scenario
# is the point at which the function is between the two cases of a wave and exponential.
#As c increases, it takes longer and longer for the block to reach its equilibrium state.
```