```
#!/bin/anaconda3/bin/python

import math
import numpy as np


#System of ode solver
# DESC: Uses the Runge-Kutta Method for Systems of Differential Equations, Chapter
5.9 Algorithm 5.7
#
# USAGE: ode_solve(func, time, initial_conditions)
#
#       func                :: A function of y and t. Returns a 1D numpy array of 1st
order diffireential equations to solve.
#       time                :: A 1D array of equally spaced time points.  Must have
more then one entry.
#       initial_conditions  :: A 1D numpy array of initial conditions for the y entry
of func.


def ode_solve(f, t, ic):

    #Assume you are given equally spaced time with more then 1 entry
    h = t[1] - t[0]

    #Store the output in a 2D array. 1st component is time, 2nd component is the
input parameters
    sol = np.zeros([len(t),len(ic)])

    #Save initial contiation to t = a
    sol[0,:] = ic

    w = ic
    i = 1
    while(i < len(t)):

                  #want to have same amount of data points as is time provided, so
dont use last time point

        k1 = h*f(w,t[i-1])
        #print("Lenths, f, w, k1, t", len(f(w,t[i-1])), len(w),len(k1),len(t))
        k2 = h*f(w + 0.5*k1, t[i-1] + h/2.0)
        k3 = h*f(w + 0.5*k2, t[i-1] + h/2.0)
        k4 = h*f(w + k3, t[i-1] + h)

        w = w + (k1 + 2.0*k2 + 2.0*k3 + k4)/6.0

        #Store solution
        sol[i,:] = w
        i = i + 1

    #End While
    return sol
#End ode_solve



#Book problem 5.9 #1c Test
#Y = [u1,u2,u3]
#def ft(Y,t):
#   dy = np.array([ Y[1], -Y[0] - 2.0*math.exp(t) + 1.0, -Y[0] - math.exp(t) + 1.0])
```

```
#   return dy
#end
#y = np.array([1.0,0.0,1.0])

#t = np.arange(0,2.0,0.5)

#sol =  ode_solve(ft, t,y)

#print("sol is ",sol)
#[[ 1.          0.          1.        ]
# [ 0.70787076 -1.24988663  0.39884862]
# [-0.33691753 -3.01764179 -0.29932294]
# [-2.41332734 -5.40523279 -0.92346873]

#And we get the books results.
```