```
"""
Registration : xxxx
Description  : Solution of first & second Order Differential Equations using Odeint
Author       : AKB
"""

import numpy as np
from scipy.integrate import quad, odeint
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Logical case switch for different problems to choose from
gquad=1; radiodec=1; dampshm=1; vanderpol=1; lorentz=1;

if(gquad):
   #=== Gauss Quadrature ==#
   def f(x) :
       return x**2
   print 'Integral 0 to 2 x^2dx using Gauss Quadrature : ', quad(f,0,2)

if(radiodec):
   #=========================================================================================#
   print    '~~~ 1ST ORDER LINEAR ODE : Radioactive Decay of Nuclear Mass ~~~'       #
   #=========================================================================================#

   k = 0.5 # parameter

   def f(x,t):
       dxdt = -k*x
       return dxdt
   t = np.linspace(0,10,100) # Creating time interval; 100 values in [0-10]
   x0 = 100                  # initial value
   sol = odeint(f, x0, t)    # solution using odeint

   # plot
   plt.figure(1)
   plt.plot(t, sol, 'ro', label='x(t)', lw=2, ms=8)
   plt.plot(t, x0*np.exp(-k*t), 'k-', label=r'$x_0 e^{-kt}$', lw=3, ms=8)
   plt.legend(loc='best', prop={'size':16})
   plt.grid()
   plt.axis([0, 10, 0, 100])
   plt.title(r'Nuclear Decay Curve $(k=0.5,x_0=100)$', fontsize=16)
   plt.xlabel('Time', fontsize = 16); plt.xticks(fontsize = 14)
   plt.ylabel('Nuclear Mass X(t)', fontsize = 16); plt.yticks(fontsize = 14)
   #plt.savefig('plot/01_radiodecay.pdf')
   plt.show()

if(dampshm):

#==========================================================================================#
   print '~~~ 2ND ORDER LINEAR ODE : Damped SHM d2x/dt2 + lambda*dx/dt + kx = 0
~~~'  #
   print '                           Forced SHM d2x/dt2 + lambda*dx/dt + kx = acos(wt)
~~~'  #

#==========================================================================================#

   k, lam = 1.0, 0.2    # Parameters for Damped Oscillation
   a, omega = 0.1, 0.1  # Parameters for Forced Oscillation

   def dshm(u,t):
       x = u[0]; y = u[1];
       dxdt = y
       dydt = -k*x - lam*y
       return np.array([dxdt, dydt])
```

```
    def fshm(u,t):
        x = u[0]; y = u[1];
        dxdt = y
        dydt = -k*x - lam*y - a*np.cos(omega*t)
        return np.array([dxdt, dydt])

    u0 = [1, 0]                  # initial values
    t = np.linspace(0,50,1000)
    sol = odeint(dshm, u0, t)    # Damped
    x1 = sol[:,0]; y1 = sol[:,1]
    sol = odeint(fshm, u0, t)    # Forced
    x2 = sol[:,0]; y2 = sol[:,1]

    # plot
    plt.figure(2)
    plt.plot(t, x1, 'k-',  label='x(t) [Damped]', lw=2, ms=6)
    plt.plot(t, y1, 'r.-', label='v(t) [Damped]', lw=2, ms=6)
    plt.plot(t, x2, 'kx',  label='x(t) [Forced]', lw=2, ms=6)
    plt.plot(t, y2, 'ro',  label='v(t) [Forced]', lw=2, ms=6)
    plt.legend(loc='best', prop={'size':16})
    plt.axis([0, 50, -1, 1])
    plt.grid()
    plt.axhline(lw=2) # draw a horizontal line
    plt.suptitle('Damped and Forced Simple Harmonic Motion',fontsize=16)
    plt.text(25,-0.3,r'$k=1,\lambda=0.2,a=\omega=0.1$', fontsize=20)
    plt.text(13,-0.65,r'(Damped) $\frac{d^2x}{dt^2}+\lambda\frac{dx}{dt}+kx=0$',
fontsize=20)
    plt.text(15,-0.9,r'(Forced) $\frac{d^2x}{dt^2}+\lambda\frac{dx}{dt}+kx=acos(\omega t)
$', fontsize=20)
    plt.xlabel('Time', fontsize = 16)
    plt.xticks(fontsize = 14)
    plt.ylabel('Displacement', fontsize = 16)
    plt.yticks(fontsize = 14)
    #plt.savefig('plot/01_dampshm.pdf')
    plt.show()

if(vanderpol):

#==================================================================================================#
    print '~~~ 2ND ORDER NONLINEAR ODE (Vanderpol Oscillator) : d2x/dt2 - mu(1-x^2)*dx/dt +
beta*x = 0 ~~~ '
    print '~~~ dydt = x/mu & dxdt = mu(1-x^3/3-beta*y); [mu & beta > 0] ~~~ '

#==================================================================================================#

    mu, beta = 0.5, 0.5   # Parameters of nonlinearity & Hookean Elasticity

    def vanderpol(X, t):
        x = X[0]; y = X[1];
        dxdt = mu*(x - x**3/3.0 - beta*y)
        dydt = x/mu
        return [dxdt, dydt]

    # main
    x0 = [1, 2]     # initial values
    t = np.linspace(0, 9000, 450)
    sol = odeint(vanderpol, x0, t)
    x = sol[:,0]; y = sol[:,1]

    # plot
    plt.figure(3)
    plt.subplot(2,1,1); # dynamics
    plt.plot(t, x, 'k-',  label='x(t)', lw=2, ms=6)
    plt.plot(t, y, 'r.-', label='y(t)', lw=2, ms=6)
    plt.legend(loc='best', prop={'size':16})
```

```
    plt.grid()
    plt.suptitle(r'Vanderpol Oscillator: $\frac{d^2x}{dt^2}+\mu(1-x^2)\frac{dx}{dt}+\beta
x=0$', fontsize=20)
    plt.title(r'$\mu='+str(mu)+', \eta='+str(beta)+'$', fontsize=20)
    plt.xlabel('Time', fontsize = 16); plt.xticks(fontsize = 14)
    plt.ylabel('Displacement', fontsize = 16); plt.yticks(fontsize = 14)

    plt.subplot(2,1,2); # phase portrait
    plt.plot(x, y,       'go-', label='x(t) vs y(t)', lw=1, ms=6)
    plt.plot(x[0], y[0], 'b*', label='Initial Value', lw=2, ms=20)
    plt.legend(loc='best', prop={'size':16})
    plt.xlabel('X Displacement', fontsize = 16); plt.xticks(fontsize = 14)
    plt.ylabel('Y Displacement', fontsize = 16); plt.yticks(fontsize = 14)
    plt.grid()
    #plt.savefig('plot/01_vanderpol.pdf')
    plt.show()

if(lorentz):
    #==================================================================================#
    print '~~~ 2ND ORDER NONLINEAR ODE : Lorentz Attractor ~~~ '                       #
    print '~~~ dx/dt=sigma*(y-x), dy/dt=x*(rho-z)-y, dz/dt=x*y-beta*z  ~~~'            #
    #==================================================================================#

    sig, rho, beta = 10.0, 28.0, 8.0/3
    def loratr(u, t):
        x,y,z = u[0],u[1],u[2]
        dxdt = sig*(y-x)
        dydt = x*(rho-z)-y
        dzdt = x*y-beta*z
        return [dxdt, dydt, dzdt]

    u0 = [0, 1.0, 0] # initial condition
    t = np.linspace(0,50,5e4)
    sol = odeint(loratr, u0, t)
    x, y, z = sol[:,0], sol[:,1], sol[:,2]

    # Plot
    plt.figure(4)
    plt.subplot(2,2,1)
    plt.plot(x, z, 'r-', label='X-Z', lw=2, ms=8)
    plt.legend(loc='best', prop={'size':16})
    plt.axis([-20, 20, 0, 50])
    plt.suptitle('Lorentz Attractor', fontsize=18)
    plt.xlabel('X', fontsize = 16)
    plt.xticks(fontsize = 14)
    plt.ylabel('Z', fontsize = 16)
    plt.yticks(fontsize = 14)

    plt.subplot(2,2,2)
    plt.plot(x, y, 'k-', label='X-Y', lw=2, ms=8)
    plt.legend(loc='best', prop={'size':16})
    plt.axis([-20, 20, -30, 30])
    plt.xlabel('X', fontsize = 16)
    plt.xticks(fontsize = 14)
    plt.ylabel('Y', fontsize = 16)
    plt.yticks(fontsize = 14)

    plt.subplot(2,2,3)
    plt.plot(y, z, 'g.', label='Y-Z', lw=2, ms=2)
    plt.legend(loc='best', prop={'size':16})
    plt.axis([-30, 30, 0, 50])
    plt.xlabel('Y', fontsize = 16)
    plt.xticks(fontsize = 14)
    plt.ylabel('Z', fontsize = 16)
    plt.yticks(fontsize = 14)
```

```
plt.text(50, 35, r'$\frac{dx}{dt}=\sigma(y-x)$', fontsize=20)
plt.text(50, 20, r'$\frac{dy}{dt}= x(\rho-z)-y$', fontsize=20)
plt.text(50, 5, r'$\frac{dz}{dt}= xy-\beta z$', fontsize=20)
#plt.savefig('plot/01_lorentz.pdf')
plt.show()
```