```
"""
CU Roll No.            :
CU Registration No. : xxxx
Description          : Basics of Scipy
Author                : AKB
"""

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

# Integer case switch for different problems to solve (feel free to add)
gquad, radiodec, simhm, dfshm, vdpol, duff, lorentz = 0, 0, 0, 0, 0, 1, 0;

if(gquad):
    #=============================================#
    print  ('~~~ Use of Gauss Quadrature ~~~')    #
    #=============================================#

    # function definition
    def f(x, a) :
        return a*x**2

    # main
    a = 1;
    print ('Integral 0 to 2 x^2dx using Gauss Quadrature : ', quad(f,0,2, args=(a,)) )
                                                 # Note the comma for single variable

if(radiodec):
    
#=============================================================================================#
    print  ('~~~ 1ST ORDER LINEAR ODE : dxdt + lambda x = 0; Radioactive Decay of Nuclear 
Mass ~~~')#
    
#=============================================================================================#

    # function definition
    def f(x,t,lam):
        dxdt = -lam*x
        return dxdt
    # Alternatively, def f(x,t,lam): return -lam*x

    # main
    x0, lam = 100, 1.0         # initial number of particles and decay constant
    t0, tf = 0, 10             # initial and final time
    t = np.linspace(t0,tf,100) # create time interval with 100 points

    sol = odeint(f, x0, t, args=(lam,))

    # plot
    plt.figure(1)
    plt.semilogy(t, sol, 'r+-', label='x(t)', lw=2, ms=8)
    plt.legend(loc='best', prop={'size':12})  # try plt.legend()
    plt.axis([0, 10, 0, 100])
    plt.title('Decay Curve', fontsize=12)
    plt.xlabel('Time', fontsize = 12); plt.xticks(fontsize = 14)
    plt.ylabel('Nuclear Mass', fontsize = 12); plt.yticks(fontsize = 14)
    #plt.savefig('plot/01_radiodecay.pdf')
    plt.show()

if(simhm):
    #==============================================================================#
    print ('~~~ 2ND ORDER LINEAR ODE : SHM d2x/dt2 + kx = 0; dxdt = y, dydt = -kx ~~~')#
    #==============================================================================#
```

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

    # main
    u0 = [1, 0]         # initial displacement and velocity
    k  = 1.0            # Restoring force parameter
    t0, tf   = 0, 50    # initial and final time
    t = np.linspace(t0,tf,1000)

    sol = odeint(shm, u0, t, args=(k,))
    x1 = sol[:,0]; y1 = sol[:,1]

    # plot
    plt.figure(2)
    plt.subplot(2,1,1)
    plt.plot(t, x1, 'ko', label='x(t)', lw=1, ms=2)
    plt.plot(t, y1, 'r+', label='v(t)', lw=1, ms=3)
    plt.legend(loc='best', prop={'size':12})
    #plt.axis([0, 50, -1, 1])
    plt.grid()
    plt.axhline(lw=2, color='coral') # draw a horizontal line
    plt.title(r'(SHM) $\frac{d^2x}{dt^2}+kx=0; $'+'k='+str(k), fontsize = 12);
    plt.xlabel('Time', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel('X(t), V(t)', fontsize = 12); plt.yticks(fontsize = 12)

    plt.subplot(2,1,2)
    plt.plot(x1, y1, 'mo-.', label='Phase Diagram', lw=2, ms=2)
    plt.plot(x1[0], y1[0], 'b*', label='Initial Value', lw=2, ms=12)
    plt.legend(loc='best', prop={'size':12})
    plt.grid()
    plt.axhline(lw=.5, color='coral') # draw a horizontal line
    plt.axvline(lw=.5, color='coral') # draw a vertical line
    plt.xlabel('X(t)', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel('V(t)', fontsize = 12); plt.yticks(fontsize = 12)

    plt.show()


if(dfshm):

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

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

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

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

    # main
```

```
    u0 = [1, 0]          # initial displacement and velocity
    k, lam   = 0.5, 0.2  # Parameters for Damped Oscillation
    a, omega = 0.1, 1.0  # Parameters for Forced Oscillation
    t0, tf   = 0, 50     # initial and final time
    t = np.linspace(t0,tf,1000)  # create time interval with 100 points

    sol = odeint(dshm, u0, t, args=(k,lam))       # Damped
    x1 = sol[:,0]; y1 = sol[:,1]
    sol = odeint(fshm, u0, t, args=(k,lam,omega)) # Forced
    x2 = sol[:,0]; y2 = sol[:,1]

    # plot
    # Damped
    plt.figure(3)
    plt.subplot(2,2,1)
    plt.plot(t, x1, 'kx',  label='x(t) [Damped]', lw=2, ms=3)
    plt.plot(t, y1, 'ro', label='v(t) [Damped]', lw=2, ms=3)
    plt.title(r'$\lambda='+str(lam)+', k='+str(k)+'$'+r', $a='+str(a)+' ,
\omega='+str(omega)+'$', fontsize=12)
    plt.legend(loc='best', prop={'size':12})
    plt.grid()
    plt.axhline(lw=2, color='gray') # draw a horizontal line
    #plt.axis([0, 50, -1, 1])
    plt.ylabel(r'$\frac{d^2x}{dt^2}+\lambda\frac{dx}{dt}+kx=0$', fontsize = 12);
plt.yticks(fontsize = 12)

    plt.subplot(2,2,2)
    plt.plot(x1, y1, 'gx', label='Phase Diagram', lw=2, ms=2)
    plt.plot(x1[0], y1[0], 'b*', label='Initial Value', lw=2, ms=12)
    plt.legend(loc='best', prop={'size':10})
    plt.grid()
    plt.axhline(lw=.5, color='coral') # draw a horizontal line
    plt.axvline(lw=.5, color='coral') # draw a vertical line
    plt.xlabel('X(t)', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel('V(t)', fontsize = 12); plt.yticks(fontsize = 12)

    # Forced
    plt.subplot(2,2,3)
    plt.plot(t, x2, 'kx',  label='x(t) [Forced]', lw=2, ms=3)
    plt.plot(t, y2, 'ro',  label='v(t) [Forced]', lw=2, ms=3)
    plt.legend(loc='best', prop={'size':12})
    plt.grid()
    plt.axhline(lw=2, color='gray') # draw a horizontal line
    #plt.axis([0, 50, -1, 1])
    plt.xlabel('Time', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel(r'$\frac{d^2x}{dt^2}+\lambda\frac{dx}{dt}+kx=acos(\omega t)$', fontsize =
12); plt.yticks(fontsize = 12)

    plt.subplot(2,2,4)
    plt.plot(x2, y2, 'mo-.', label='Phase Diagram', lw=2, ms=2)
    plt.plot(x2[0], y2[0], 'b*', label='Initial Value', lw=2, ms=12)
    plt.legend(loc='best', prop={'size':10})
    plt.grid()
    plt.axhline(lw=.5, color='coral') # draw a horizontal line
    plt.axvline(lw=.5, color='coral') # draw a vertical line
    plt.xlabel('X(t)', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel('V(t)', fontsize = 12); plt.yticks(fontsize = 12)
    #plt.savefig('plot/01_dampshm.pdf')

    plt.show()

if(vdpol):
#==========================================================================================
    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] ~~~ ')

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

    # function definition
    def vanderpol(X, t, mu, beta):
        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
    mu, beta = 0.1, 0.2  # Parameters of nonlinearity & Hookean Elasticity
    t0, tf = 0, 200      # initial and final time
    t = np.linspace(t0,tf,1000)  # create time interval with 1000 points

    sol = odeint(vanderpol, x0, t, args=(mu, beta))
    x, y = sol[:,0], sol[:,1]

    # plot
    plt.figure(4)
    plt.subplot(2,1,1); # dynamics
    plt.plot(t, x, 'kx-',  label='x(t)', lw=.5, ms=2)
    plt.plot(t, y, 'ro-',  label='v(t)', lw=.5, ms=2)
    plt.legend(loc='best', prop={'size':12})
    plt.grid()
    plt.title(r'Vanderpol Oscillator: $\frac{d^2x}{dt^2}+\mu(1-x^2)\frac{dx}{dt}+\beta x=0;
\mu='+str(mu)+'$'+r'$, \beta='+str(beta)+'$', fontsize=12)
    plt.xlabel('Time', fontsize = 12); plt.xticks(fontsize = 14)
    plt.ylabel('X(t),V(t)', fontsize = 12); plt.yticks(fontsize = 14)

    plt.subplot(2,1,2); # phase portrait
    plt.plot(x, y,       'mo-.', label='x(t) vs y(t)', lw=1, ms=2)
    plt.plot(x[0], y[0], 'b*', label='Initial Value', lw=2, ms=12)
    plt.legend(loc='best', prop={'size':12})
    plt.xlabel('X(t)', fontsize = 12); plt.xticks(fontsize = 14)
    plt.ylabel('V(t)', fontsize = 12); plt.yticks(fontsize = 14)
    plt.grid()
    #plt.savefig('plot/01_vanderpol.pdf')
    plt.show()


if(duff):

#=======================================================================================================
    print ('~~~ 2ND ORDER NONLINEAR ODE : d2x/dt2 + lambda*dx/dt + kx + alpha x^3 =
acos(wt); dxdt = y, dydt = -kx-alpha x^3-lambda y + acos(wt)  ~~~')#

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

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

    # main
    u0 = [1, 0]          # initial displacement and velocity
    k, lam, alpha, a, omega = 0.5, 0.1, 1.0, 0.2, 1.0  # Parameters
    t0, tf   = 0, 50     # initial and final time
    t = np.linspace(t0,tf,1000)  # create time interval with 100 points

    sol = odeint(duffing, u0, t, args=(k,lam,alpha,a,omega)) # Forced
    x = sol[:,0]; y = sol[:,1]
```

```
    # plot
    plt.figure(5)
    plt.subplot(2,1,1)
    plt.plot(t, x, 'kx',  label='x(t) [Duffing]', lw=2, ms=2)
    plt.plot(t, y, 'ro',  label='v(t) [Duffing]', lw=2, ms=2)
    plt.legend(loc='best', prop={'size':12})
    plt.grid()
    plt.axhline(lw=2, color='gray') # draw a horizontal line
    #plt.axis([0, 50, -1, 1])
    plt.text(15,-0.7, r'$\lambda='+str(lam)+', k='+str(k)+'$'+r' ,$\alpha='+str(alpha)
+' ,a='+str(a)+' ,\omega='+str(omega)+'$', fontsize=12)
    plt.xlabel('Time', fontsize = 12); plt.xticks(fontsize = 12)
    plt.title(r'$\frac{d^2x}{dt^2}+\lambda\frac{dx}{dt}+kx+\alpha x^3=acos(\omega t)$',
fontsize = 12);
    plt.ylabel('X(t), V(t)', fontsize=12); plt.yticks(fontsize = 12)

    plt.subplot(2,1,2)
    plt.plot(x, y, 'mo-.', label='Phase Diagram', lw=2, ms=2)
    plt.plot(x[0], y[0], 'b*', label='Initial Value', lw=2, ms=12)
    plt.legend(loc='best', prop={'size':10})
    plt.grid()
    plt.axhline(lw=.5, color='coral') # draw a horizontal line
    plt.axvline(lw=.5, color='coral') # draw a vertical line
    plt.xlabel('X(t)', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel('V(t)', fontsize = 12); plt.yticks(fontsize = 12)
    #plt.savefig('plot/01_dampshm.pdf')

    plt.show()

if(lorentz):

#===============================================================================================================
    print ('~~~ 2ND ORDER ODE : Lorentz Attractor ~~~ ')
    print ('~~~ dx/dt=sigma*(y-x), dy/dt=x*(rho-z)-y, dz/dt=x*y-beta*z  ~~~')
    print ('A 2D fluid-cell when heated from underneath and cooled from above (like earths
atmosphere) creates convection.')
    print ('x = rate of convective overturning, y = horizontal and z = vertical temperature
variations')

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

    # function definition
    def loratr(u, t, sig, rho, beta):
        x,y,z = u[0],u[1],u[2]
        dxdt = sig*(y-x)
        dydt = x*(rho-z)-y
        dzdt = x*y-beta*z
        return np.array([dxdt, dydt, dzdt])
    # alternatively, def loratr(u, t, sig, rho, beta):
    #                    return [sig*(u[1]-u[0]), u[0]*(rho-u[2])-u[1], u[0]*u[1]-
beta*u[2]]

    # main
    u0 = [0, 1.0, 0]                    # initial x, y, z
    sig, rho, beta = 10.0, 26.0, 8.0/3  # parameters
    t0, tf = 0, 100                     # initial and final time
    t = np.linspace(t0,tf,5000)         # create time interval with 450 points

    sol = odeint(loratr, u0, t, args=(sig, rho, beta))
    x, y, z = sol[:,0], sol[:,1], sol[:,2]

    # plot X(t), Y(t), Z(t)
    plt.figure(6)
    plt.subplot(3,1,1);
    plt.plot(t, x, 'r-', label='X(t)', lw=1, ms=2)
```

```
    plt.xticks(fontsize = 12); plt.ylabel('X', fontsize = 12); plt.yticks(fontsize = 12);
plt.grid();

    plt.subplot(3,1,2);
    plt.plot(t, y, 'g-', label='Y(t)', lw=1, ms=2)
    plt.xticks(fontsize = 12); plt.ylabel('Y', fontsize = 12); plt.yticks(fontsize = 12);
plt.grid();

    plt.subplot(3,1,3);
    plt.plot(t, z, 'b-', label='Z(t)', lw=1, ms=2)
    plt.xlabel('t', fontsize=14); plt.xticks(fontsize = 12); plt.ylabel('Z', fontsize =
12); plt.yticks(fontsize = 12); plt.grid();

    # plot X-Y, Y-Z, Z-X
    plt.figure(7)
    plt.subplot(2,2,1)
    plt.plot(x, z, 'r-', label='X-Z', lw=.5, ms=2)
    plt.legend(loc='best', prop={'size':12})
    plt.suptitle('Lorentz Attractor', fontsize=14)
    plt.xlabel('X', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel('Z', fontsize = 12); plt.yticks(fontsize = 12)

    plt.subplot(2,2,2)
    plt.plot(x, y, 'k-', label='X-Y', lw=.5, ms=2)
    plt.legend(loc='best', prop={'size':12})
    plt.xlabel('X', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel('Y', fontsize = 12); plt.yticks(fontsize = 12)

    plt.subplot(2,2,3)
    plt.plot(y, z, 'g-.', label='Y-Z', lw=.5, ms=2)
    plt.legend(loc='best', prop={'size':12})
    plt.xlabel('Y', fontsize = 12); plt.xticks(fontsize = 12)
    plt.ylabel('Z', fontsize = 12); plt.yticks(fontsize = 12)

    plt.text(50, 35, r'$\frac{dx}{dt}=\sigma(y-x)$',  fontsize=12)
    plt.text(50, 20, r'$\frac{dy}{dt}= x(\rho-z)-y$', fontsize=12)
    plt.text(50, 5, r'$\frac{dz}{dt}= xy-\beta z$',   fontsize=12)
    #plt.savefig('plot/01_lorentz.pdf')
    plt.show()
```