UNIT 6 Runge Kutta 2 Method to Solve ODE

1. To Solve First Order Differential Equation by RK2 Method and compare it with Exact Solution and Inbuilt Function.

import libraries

```
for i in range(len(x_values)-1):
import numpy as np
from scipy.integrate import odeint
                                                         k1=h*f(y_values[i],x_values[i])
                                                         k2=h*f(y_values[i]+k1,x_values[i]+h)
import matplotlib.pyplot as plt
                                                         delY=0.5*(k1+k2)
def f(y,x):
              #Eqution to solve
  return np.exp(x)
                                                         y_values[i+1]=y_values[i]+delY
def f_exact(y,x):
                                                       y_exact=f_exact(y_exact,x_values)
                                                       y_odeint=odeint(f,y0,x_values)
  return np.exp(x)
                                                       plt.subplot(1,3,1)
print("\n\n\tMehendi
Hasan\n\t2230248\n\n"
                                                       plt.plot(x_values,y_values,label="RK2")
                                                       plt.xlabel("X Points")
#initial values
                                                       plt.ylabel("Y Points")
x0=float(input("Enter Initial value of X: "))
                                                       plt.legend()
y0=float(input("Enter Value of Y at Initial
                                                       plt.subplot(1,3,2)
value of X: "))
                                                       plt.plot(x_values,y_exact,label="Exact
h=float(input("Enter Step Size: "))
                                                       Solution")
b=float(input("Enter last value of interval: "))
                                                       plt.xlabel("X Points")
x_values=np.arange(0,b+h,h)
                                                       plt.ylabel("Y Points")
y_values=np.zeros(len(x_values))
                                                       plt.legend()
y_exact=np.zeros(len(x_values))
                                                       plt.subplot(1,3,3)
y_odeint=np.zeros(len(x_values))
                                                       plt.plot(x_values,y_odeint,label="Odeint
x_values[0]=x0
                                                       solution")
y_values[0]=y0
                                                       plt.xlabel("X Points")
y_odeint[0]=y0
                                                       plt.ylabel("Y Points")
y exact[0]=y0
                                                       plt.legend()
                                                       plt.show()
```

2. To Plot Newton's cooling law ODE by RK2 method, Exact solution & Inbuilt solver.

#libraries

import matplotlib.pyplot as plt

import numpy as np

import scipy.integrate as it

def f(T,t): # Differential Equation of cooling

return (-K)*(T-Ts)

def fExact(T,t):

Ts + ((T0-Ts)*np.etp((-K)*(t)))

Exact Equation of cooling

print("\n\n\tMehendi
Hasan\n\n\t2230248\n\n")

print("Newton's Law of Cooling\n\nTemperature is in Degree Celsius and time is in secons\n\n")

T0=int(input("Enter initial Temperature of Object: "))

Ts=int(input("Enter Surrounding temperature: "))

t=int(input("Enter time from t=0,
at which temperature of Object
to be calculated: "))

h=0.001 #step size

K=0.1 #cooling constant

t values=np.arange(0,t+h,h)

T_values=np.zeros(len(t_values))

T_Exact=np.zeros(len(t_values))

T_odeint=np.zeros(len(t_values))

t_values[0]=0

T values[0]=T0

T_odeint[0]=T0

T_Exact[0]=T0

for i in range(len(t_values)-1):

k1=h*f(T_values[i],t_values[i])

k2=h*f(T_values[i]+k1,t_values[i] +h)

delY=0.5*(k1+k2)

T_values[i+1]=T_values[i]+delY

T_Exact = Ts + ((T0-Ts)*np.exp((-K)*(t_values))) #solution equation

T_odeint=it.odeint(f,T0,t_values)
#odeint solution

#ploting

plt.subplot(3,1,1)

```
plt.ylabel("Temperature of
plt.plot(t values,T values,label="
RK2 Solution",color="red")
                                         Object")
                                         plt.subplot(3,1,3)
plt.grid()
plt.title("RK2 Method")
                                         plt.plot(t values,T odeint,label='
                                         Odient Solution',color="orange")
plt.xlabel("Time")
                                         plt.grid()
plt.ylabel("Temperature of
Object")
                                         plt.title("Odeient Solution")
plt.legend()
                                         plt.xlabel("Time")
                                         plt.ylabel("Temperature of
plt.subplot(3,1,2)
                                         Object")
plt.plot(t values,T Exact,label='E
xact Equation
                                         plt.legend()
Solution',color="blue")
                                         plt.suptitle("Mehendi Hasan
                                         B.SC.(H) Physics\nTo Plot
plt.grid()
                                         Newton's cooling law ODE by RK2
plt.title("Exact Equation")
                                         method, Exact solution & Inbuilt
plt.xlabel("Time")
                                         solver")
                                         plt.show()
```

3. To Plot Radioactive Decay ODE by RK2 method, Exact solution & Inbuilt solver.

#importing libraries

import matplotlib.pyplot as plt import numpy as np import scipy.integrate as it

def diff_Equ(N,t): #dfferential equation of Radio activee decay

return (-1)*(K)*(N) # K is the decay constant and N is the number of parent atoms at time instant any

def Exact_Equ(N,t): #solution
equation of Radio activee decay

return N0*(np.exp((-1)*(K)*(t)))
N0 is the number of parent
atoms at t=0

print("\n\n\tMehendi
Hasan\n\n\t2230248\n\nRadioac
tive Decay \n\nTime is in
Seconds\n")

#itaking input from user

N0=int(input("Enter Number of Parent Atoms at t=0: "))

t=int(input("Enter time instant at
which Remaining of Parent Atoms
to be calculated: "))

K=float(input("Enter Radioactive Decay constant value: ")) #
Radioactive Decay Constant

h=0.001 # Step size

t_array=np.arange(0,t+h,h)
#initializing time
array(independent Variable)

Y_differential=np.zeros(len(t_arra y)) #Initializing array for Y_differential of dependent variable

Y_Exact=np.zeros(len(t_array))
#Initializing array for
Y_differential of dependent
variable(Solution equation)

Y_differential[0] = Y_Exact[0] = N0 #Initial Y_differential of

dependent variable Y at independent variable t=0

for i in range(len(t_array)-1):
k1=h*diff_Equ(Y_differential[i],t_
array[i])

k2=h*diff_Equ(Y_differential[i]+k
1,t_array[i]+h)
 delY=0.5*(k1+k2)

Y_differential[i+1]=Y_differential[i]+delY

Y_Exact=Exact_Equ(Y_Exact,t_arr ay) # solution equation

#ploting all the Y_differential of dependent variable with respect to independent variable

plt.subplot(3,1,1)

plt.plot(t_array,Y_differential,col
or="green",label="RK2's
Solution")

plt.grid()

plt.xlabel("Time (Second)") plt.ylabel("No. of parent Atoms") plt.legend() plt.subplot(3,1,2) plt.plot(t array,Y Exact,color="re d",label='Exact Equation Solution') plt.grid() plt.xlabel("Time (Second)") plt.ylabel("No. of parent Atoms") plt.legend() plt.subplot(3,1,3)plt.plot(t array,solOdeint,color=" blue", label='Odeint Solution') plt.grid() plt.xlabel("Time (Second)") plt.ylabel("No. of parent Atoms") plt.legend() plt.suptitle("Mehendi Hasan B.SC.(H) Physics 2230248\nTo Plot Radioactive Decay ODE by RK2 method, Exact solution & Inbuilt solver.")

plt.show()

4. To Plot Charging and Discharging of a capacitor in RC circuit ODE with DC source by RK2 Method, Exact solution, Inbuilt solver.

#importing libraries

import matplotlib.pyplot as plt import numpy as np import scipy.integrate as it

def diff_equ_charging(q,t):
Differential Equation of
Charging

return ((C*E - q)/(R*C))

def Exact_equ_charging(t):
#Solution equation of
Differential Equation of
Charging

return (C*E)*(1-(np.exp(((-1)*t)/(R*C))))

def diff_equ_discharging(q,t):

Differential Equation of Discharging

return ((-1)*q)/(R*C)

def Exact_equ_discharging(t):

#Solution equation of Differential Equation of Discharging

return ((C*E)*(np.exp(((-1)*t)/(R*C))))

print("\n\n\tMehendi
Hasan\n\n\t2230248\n\nRC
Circuit Charging and
Discharging of Capacitor\n\n")

print("Capacitance is in Farad,
resistance is in ohm,time is in
second,charge in
coulomb,voltage in volts.\n\n")

#taking inputs from user for the terms envoled in equations

C=float(input("Enter Capacitance of Capacitor: "))

E=float(input("Enter EMF of
Battery: "))

R=float(input("Enter Resistance of Resistor: "))

t=float(input("Enter time
instant at which charge on
capacitor to be calculated: "))

h=0.1 #Step size

Qmax=C*E #max value of charge on capacitor

t_array=np.arange(0,t+h,h)
#initializing time
array(independent Variable)

#Charging of Capacitor

Y_diff_charging=np.zeros(len(t_ array)) #Initializing array for values of dependent variable(RK2's method)

Y_Exact_charging=np.zeros(len(t_array)) #Initializing array for values of dependent variable(Solution equation)

Y_Exact_charging[0] = Y_diff_charging[0] = 0 #Initial values of dependent variable Y at dependent variable t=0

for i in range(len(t_array)-1):
k1=h*diff_equ_charging(Y_diff
_charging[i],t_array[i])

k2=h*diff_equ_charging(Y_diff
_charging[i]+k1,t_array[i]+h)

delY=0.5*(k1+k2)

Y_diff_charging[i+1]=Y_diff_charging[i]+delY

Y_Exact_charging=Exact_equ_c harging(t_array)
#Solution Equation

solOdeintCharging=it.odeint(dif f_equ_charging,Y_diff_charging [0],t_array) #Odeint solution

#Discharging of Capacitor

Y_diff_discharging=np.zeros(len (t_array)) #Initializing array for values of dependent variable(RK2's method)

Y_Exact_discharging=np.zeros(len(t_array)) #Initializing array for values of dependent variable(Solution equation)

Y_Exact_discharging[0] =
Y_diff_discharging[0] = Qmax
#Initial values of dependent
variable Y at independent
variable t=0

plt.xlabel("Time") for i in range(len(t_array)-1): plt.ylabel("Charge at Capacitor") k1=h*diff equ discharging(Y d iff discharging[i],t array[i]) plt.title("RK2's Solution of Charging") k2=h*diff equ discharging(Y d plt.legend() iff_discharging[i]+k1,t_array[i]+ plt.subplot(3,2,4) h) plt.plot(t array,Y Exact chargi delY=0.5*(k1+k2)ng,color='red',label="Charge") Y diff discharging[i+1]=Y diff plt.grid() discharging[i]+delY plt.xlabel("Time") plt.ylabel("Charge at Y Exact discharging=Exact equ Capacitor") discharging(t array) plt.title("Exact Equation of **#Solution Equation** Charging") solOdeintDischarging=it.odeint(plt.legend() diff equ discharging, Y diff dis charging[0],t array) plt.subplot(3,2,1)**#Odeint solution** plt.plot(t array,Y diff dischargi ng,color='orange',label="Charg e") #ploting all the values of dependent variable with plt.grid() respect to independent plt.xlabel("Time") variable plt.ylabel("Charge at plt.subplot(3,2,2)Capacitor") plt.plot(t array,Y diff charging plt.title("RK2's Solution of ,color='blue',label="Charge") Discharging") plt.grid()

```
plt.legend()
                                       plt.legend()
                                       plt.subplot(3,2,5)
plt.subplot(3,2,3)
plt.plot(t array,Y Exact discha
                                       plt.plot(t array,solOdeintDisch
rging,color='green',label="Char
                                       arging,color='red',label="Charg
ge")
                                       e")
plt.grid()
                                       plt.grid()
                                       plt.xlabel("Time")
plt.xlabel("Time")
                                       plt.ylabel("Charge at
plt.ylabel("Charge at
Capacitor")
                                       Capacitor")
plt.title("Exact Equation of
                                       plt.title("Odeint Solution of
Discharging")
                                       Discharging")
plt.legend()
                                       plt.legend()
plt.subplot(3,2,6)
                                       plt.suptitle("Mehendi Hasan
                                       B.SC.(H) Physics 2230248\nTo
plt.plot(t array,solOdeintChargi
                                       Plot Charging and Discharging
ng,color='orange',label="Charg
                                       of a capacitor in RC circuit ODE
e")
                                       with DC source by RK2 Method,
plt.grid()
                                       Exact solution, Inbuilt solver")
plt.xlabel("Time")
                                       plt.show()
plt.ylabel("Charge at
Capacitor")
plt.title("Odeint Solution of
Charging")
```

5. To Plot Current in RC circuit and potential ODE with DC source by RK2 Method, Exact solution, Inbuilt solver.

#importing libraries to be used

import matplotlib.pyplot as plt import numpy as np import scipy.integrate as it

class RC: # Created a class of RC Circuit which have multiple Functions

def current(I,t): # Current
v/s time graph using RK2's
Method

return ((-1)*(I))/(R*C)

def current_exact(t): #
Current v/s time graph by
ploting the solution equation
of ODE

return IO*(np.exp((-1)*t/(R*C)))

def Vr(Vr,t): #

Voltage across resistor v/s time graph using RK2's Method

return -Vr/(R*C)

def VrExact(t): # Voltage across resistor v/s time graph by ploting the solution equation of ODE

return V*(np.exp((t)/(R*C)))

def Vc(Vc,t): #
Voltage across capacitor v/s
time graph using RK2's Method

return (1/(R*C))*(V-Vc)

def VcExact(t): #
Voltage across capacitor v/s
time graph by ploting the
solution equation of ODE

return V*(1-(np.exp((t)/(R*C))))

print("\n\n\tMehendi
Hasan\n\n\t2230248\n\nRC
Circuit\n\n")

print("Capacitance is in Farad,
resistance is in ohm,time is in
second,charge in
coulomb,voltage in volts.\n\n")

input constant values

R=float(input('Enter the value of resistance in ohms:'))

#resistance

C=float(input('Enter the value of capacitance in farads:')) # capacitance

V=float(input('Enter the value of EMF in volts:')) # EMF

T fin=float(input('Enter time instant at which current to be # time instant measured:'))

h=0.001 #step size

time_array=np.arange(0,T_fin+ # X-coordinate (time) h,h)

Current v/s time

IO=V/R #current in circuit at t=0

yPointsCurrent=np.zeros(len(ti me array))

yPointsCurrentExact=np.zeros(I en(time array))

yPointsCurrent[0]=I0

for i in range(len(time array)-1):

k1=h*RC.current(yPointsCurren t[i],time array[i]) k2=h*RC.current(yPointsCurren

t[i]+k1,time array[i]+h)

delY=0.5*(k1+k2)

yPointsCurrent[i+1]=yPointsCur rent[i]+delY

yPointsCurrentExact=RC.curren t exact(time array)

Solution Equation

solOdeintYPointsCurrent=it.ode int(RC.current,I0,time array) #odeint solution

Voltage across resistor v/s time

Vr0=V

yPointsVr=np.zeros(len(time ar ray))

yPointsVrExact=np.zeros(len(ti me array))

yPointsVr[0]=Vr0

for i in range(len(time array)-1):

k1=h*RC.Vr(yPointsVr[i],time a rray[i])

k2=h*RC.Vr(yPointsVr[i]+k1,tim e_array[i]+h)

delY=0.5*(k1+k2)

yPointsVr[i+1]=yPointsVr[i]+del Υ

yPointsVrExact=RC.VrExact(tim solOdeintYPointsVc=it.odeint(R # Solution e array) C.Vc,Vc0,time array) **Equation** #odeint solution solOdeintYPointsVr=it.odeint(R C.Vr,Vr0,time array) # plot of I v/s t #odeint solution plt.subplot(3,2,1)plt.plot(time array,yPointsCurr # Voltage across capacitor v/s ent,color='red',label="I") time plt.xlabel('Time(s)') Vc0=0plt.ylabel('Current(amps)') yPointsVc=np.zeros(len(time ar plt.title("Current v/s time ray)) RK2's") yPointsVcExact=np.zeros(len(ti plt.grid('true') me_array)) plt.legend() vPointsVc[0]=Vc0 plt.subplot(3,2,2) for i in range(len(time array)-1): plt.plot(time array,yPointsCurr k1=h*RC.Vc(yPointsVc[i],time entExact, color='blue',label="I") array[i]) plt.xlabel('Time(s)') plt.ylabel('Current(amps)') k2=h*RC.Vc(yPointsVc[i]+k1,tim plt.title("Current v/s time e array[i]+h) Solution Equation") delY=0.5*(k1+k2)plt.grid('true') yPointsVc[i+1]=yPointsVc[i]+del plt.legend() Υ # plot of Vr v/s t yPointsVcExact=RC.VcExact(tim e array) # Solution Equation

plt.subplot(3,2,3)

```
plt.plot(time array,solOdeintYP
plt.plot(time array,yPointsVr,c
                                       ointsVc, color='red',label="Vc")
olor='red',label="Vr")
plt.plot(time array,yPointsVc,c
                                       plt.xlabel('Time(s)')
olor='blue',label="Vc")
                                       plt.ylabel("(volts)")
plt.xlabel('Time(s)')
                                       plt.title("Vr and Vc v/s time
plt.ylabel('(volts)')
                                       Odeint Solution")
plt.title("Vr and Vc v/s time
                                       plt.grid('true')
RK2s ")
                                       plt.legend()
plt.grid('true')
                                       plt.subplot(3,2,6)
plt.legend()
                                       plt.plot(time_array,solOdeintYP
                                       ointsCurrent,
plt.subplot(3,2,4)
                                       color='blue',label="Current")
plt.plot(time array,yPointsVrEx
act, color='blue', label="Vr")
                                       plt.xlabel('Time(s)')
                                       plt.ylabel("(volts)")
plt.plot(time array,yPointsVcEx
act, color='red',label="Vc")
                                       plt.title("Current v/s time
plt.xlabel('Time(s)')
                                       Odeint Solution equation")
plt.ylabel("(volts)")
                                       plt.grid('true')
plt.title("Vr and Vc v/s time
                                       plt.legend()
Solution equation")
                                       plt.suptitle("Mehendi Hasan
plt.grid('true')
                                       B.SC.(H) Physics 2230248\nTo
                                       Plot Current in RC circuit and
plt.legend()
                                       potential ODE with DC source
                                       by RK2 Method, Exact solution,
                                       Inbuilt solver.")
plt.subplot(3,2,5)
                                       plt.show()
plt.plot(time array,solOdeintYP
ointsVr, color='blue',label="Vr")
```

6. To Plot Current in RL circuit ODE with DC source by RK2 Method, Exact solution, Inbuilt solver.

#importing libraries to be used

import matplotlib.pyplot as plt import numpy as np import scipy.integrate as it

def diffEquation(i,t):
 return (V/L)-((R*i)/L)

def solEquation(i,t):
 return (V/R)*(1-(np.exp((((-1)*R)*t)/L)))

print("\n\n\tMehendi
Hasan\n\n\t2230248\n\nVariat
ion of curent with time in RL
Circuit \n\n")

print("Resistance is in ohm,time
is in second,Inductance in
henry,voltage in volts.\n\n")

#taking inputs from user for the
terms envoled in equations
L=float(input("Enter Inductance
of Inductor: "))

V=float(input("Enter EMF of Battery: "))

R=float(input("Enter Resistance of Resistor: "))

t=float(input("Enter time instant at which Current through inductor to be calculated: "))

h=0.001 #step size

time array=np.arange(0,t+h,h)

Current v/s time

X-coordinate (time)

I0=0 #current in circuit at t=0
yPointsCurrent=np.zeros(len(ti
me_array))
yPointsCurrentExact=np.zeros(l
en(time_array))
yPointsCurrent[0]=I0
yPointsCurrentExact[0]=I0

```
for i in range(len(time array)-
                                      plt.legend()
1):
                                      plt.subplot(1,3,2)
yPointsCurrent[i+1]=yPointsCur
                                      plt.plot(time array,yPointsCurr
rent[i]+h*diffEquation(yPointsC
                                      entExact, color='blue',label="I")
urrent[i],time array[i])
#RK2's Method
                                      plt.xlabel('Time(s)')
k1=h*diffEquation(yPointsCurr
                                      plt.ylabel("Current(Ampere)")
ent[i],time array[i])
                                      plt.title("Current v/s time
k2=h*diffEquation(yPointsCurr
                                      Solution Equation")
ent[i]+k1,time array[i]+h)
                                      plt.grid('true')
  delY=0.5*(k1+k2)
                                      plt.legend()
yPointsCurrent[i+1]=yPointsCur
                                      plt.subplot(1,3,3)
rent[i]+delY
                                      plt.plot(time array,solOdeintYP
yPointsCurrentExact=solEquati
                                      ointsCurrent,
on(yPointsCurrentExact,time a
                                      color='green',label="Current")
          # Solution Equation
rray)
                                      plt.xlabel('Time(s)')
solOdeintYPointsCurrent=it.ode
int(diffEquation,I0,time array)
                                      plt.ylabel("Current(Ampere)")
#odeint solution
                                      plt.title("Current v/s time
# plot of I v/s t
                                      Odeint Solution equation")
plt.subplot(1,3,1)
                                      plt.grid('true')
plt.plot(time array,yPointsCurr
                                      plt.suptitle("Mehendi Hasan
ent,color='red',label="I RK2")
                                      B.SC.(H) Physics 2230248\nTo
                                      Plot Current in RL circuit ODE
plt.xlabel('Time(s)')
                                      with DC source by RK2 Method,
plt.ylabel("Current(Ampere)")
                                      Exact solution, Inbuilt solver.")
plt.title("Current v/s time
                                      plt.legend()
RK2's")
                                      plt.show()
plt.grid('true')
```