UNIT 6 RUNGE KUTTA 4 Method for Solving ODE

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

```
# import libraries
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def f(y,x):        #Eqution to solve
    return np.exp(x)
def f_exact(y,x):
    return np.exp(x)

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

#initial values
x0=float(input("Enter Initial value of X: "))
y0=float(input("Enter Value of Y at Initial value of X: "))
h=float(input("Enter Step Size: "))
b=float(input("Enter last value of interval: "))
x_values=np.arange(0,b+h,h)
y_values=np.zeros(len(x_values))
y_exact=np.zeros(len(x_values))
y_odeint=np.zeros(len(x_values))
x_values[0]=x0
y_values[0]=y0
```

```
y_odeint[0]=y0
y_exact[0]=y0
for i in range(len(x_values)-1):

k1=h*f(y_values[i],x_values[i])

k2=h*f(y_values[i]+(k1/2),x_values[i]+(h/2))

k3=h*f(y_values[i]+(k2/2),x_values[i]+(h/2))

k4=h*f(y_values[i]+(k3),x_values[i]+(h))

delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)

y_values[i+1]=y_values[i]+delY
y_exact=f_exact(y_exact,x_values)
y_odeint=odeint(f,y0,x_values)

plt.subplot(1,3,1)
plt.plot(x_values,y_values,label="RK2")
plt.xlabel("X Points")
plt.ylabel("Y Points")
plt.legend()

plt.subplot(1,3,2)
plt.plot(x_values,y_exact,label="Exact Solution")
plt.xlabel("X Points")
plt.ylabel("Y Points")
plt.legend()

plt.subplot(1,3,3)
plt.plot(x_values,y_odeint,label="Odeint solution")
plt.xlabel("X Points")
plt.ylabel("Y Points")
plt.legend()
plt.show()
```

2. To Plot Newton's cooling law ODE by RK4 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 seconds\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]+(k1/2),t_values[i]+(h/2))
k3=h*f(T_values[i]+(k2/2),t_values[i]+(h/2))
k4=h*f(T_values[i]+(k3),t_values[i]+(h))
delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)
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.plot(t_values,T_values,label="Temperature",color="red")
plt.grid()
plt.title("RK4 Method")
plt.xlabel("Time-->")
plt.ylabel("Temperature of Object-->")
plt.legend()
plt.subplot(3,1,2)
plt.plot(t_values,T_Exact,label='Temperature',color="blue")
plt.grid()
plt.title("Exact Equation")
plt.xlabel("Time-->")
plt.ylabel("Temperature of Object-->")
plt.legend()
plt.subplot(3,1,3)
plt.plot(t_values,T_odeint,label='Temperature',color="orange")
plt.grid()
plt.title("Odeient Solution")
plt.xlabel("Time-->")
plt.ylabel("Temperature of Object-->")
plt.legend()
plt.suptitle("Mehendi Hasan B.SC.(H) Physics\nTo Plot Newton's cooling law ODE by RK4 method, Exact solution & Inbuilt solver")
plt.show()
```

3. To Plot Radioactive Decay ODE by RK4 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 activedecay**

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 activedecay**

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\nRadio active 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)

Y_differential=np.zeros(len(t_array))

Y_Exact=np.zeros(len(t_array))

Y_differential[0] = Y_Exact[0] = N0

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]+(k1/2),t_array[i]+(h/2))

```
k3=h*diff_Equ(Y_differential[i]+(k2/2),t_array[i]+(h/2))
k4=h*diff_Equ(Y_differential[i]+(k3),t_array[i]+(h))
delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)
Y_differential[i+1]=Y_differential[i]+delY


Y_Exact=Exact_Equ(Y_Exact,t_array)
# solution equation
solOdeint=it.odeint(diff_Equ,N0,t_array)
# odeint solution


#ploting
plt.subplot(3,1,1)
plt.plot(t_array,Y_differential,color="green",label="Number of Parent Atoms")
plt.title("RK4'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="red",label='Number of Parent Atoms')
plt.title("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='Number of Parent Atoms')
plt.title("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 RK4 method, Exact solution & Inbuilt solver.")
plt.show()
```

4. To Plot Charging and Discharging of a capacitor in RC circuit ODE with DC source by RK4 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=0.01      #Step size

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

```
t_array=np.arange(0,t+h,h)
```

#Charging of Capacitor

```
Y_diff_charging=np.zeros(len(t_array))

Y_Exact_charging=np.zeros(len(t_array))

Y_Exact_charging[0] = Y_diff_charging[0] = 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]+(k1/2),t_array[i]+(h/2))

k3=h*diff_equ_charging(Y_diff_charging[i]+(k2/2),t_array[i]+(h/2))

k4=h*diff_equ_charging(Y_diff_charging[i]+(k3),t_array[i]+(h))

delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)

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


Y_Exact_charging=Exact_equ_charging(t_array)
```

#Solution Equation

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

#Discharging of Capacitor

```
Y_diff_discharging=np.zeros(len(t_array))

Y_Exact_discharging=np.zeros(len(t_array))

Y_Exact_discharging[0] = Y_diff_discharging[0] = Qmax

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

k1=h*diff_equ_discharging(Y_diff_discharging[i],t_array[i])
k2=h*diff_equ_discharging(Y_diff_discharging[i]+(k1/2),t_array[i]+(h/2))
k3=h*diff_equ_discharging(Y_diff_discharging[i]+(k2/2),t_array[i]+(h/2))
k4=h*diff_equ_discharging(Y_diff_discharging[i]+(k3),t_array[i]+(h))

delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)

Y_diff_discharging[i+1]=Y_diff_discharging[i]+delY


Y_Exact_discharging=Exact_equ_discharging(t_array)
```
#Solution Equation

```
solOdeintDischarging=it.odeint(diff_equ_discharging,Y_diff_discharging[0],t_array)
```
#Odeint solution

#ploting

```
plt.subplot(3,2,2)

plt.plot(t_array,Y_diff_charging,color='blue',label="Charge")

plt.grid()
```

```
plt.xlabel("Time")
plt.ylabel("Charge at Capacitor")
plt.title("RK4's Solution of Charging")
plt.legend()
plt.subplot(3,2,4)
plt.plot(t_array,Y_Exact_charging,color='red',label="Charge")
plt.grid()
plt.xlabel("Time")
plt.ylabel("Charge at Capacitor")
plt.title("Exact Equation of Charging")
plt.legend()
plt.subplot(3,2,1)
plt.plot(t_array,Y_diff_discharging,color='orange',label="Charge")
plt.grid()
plt.xlabel("Time")
plt.ylabel("Charge at Capacitor")
plt.title("RK4's Solution of Discharging")
plt.legend()
plt.subplot(3,2,3)
plt.plot(t_array,Y_Exact_discharging,color='green',label="Charge")
plt.grid()
plt.xlabel("Time")
plt.ylabel("Charge at Capacitor")
plt.title("Exact Equation of Discharging")
plt.legend()
plt.subplot(3,2,6)
plt.plot(t_array,solOdeintCharging,color='orange',label="Charge")
plt.grid()
plt.xlabel("Time")
plt.ylabel("Charge at Capacitor")
plt.title("Odeint Solution of Charging")
plt.legend()
plt.subplot(3,2,5)
plt.plot(t_array,solOdeintDischarging,color='red',label="Charge")
plt.grid()
plt.xlabel("Time")
plt.ylabel("Charge at Capacitor")
plt.title("Odeint Solution of Discharging")
plt.legend()
plt.suptitle("Mehendi Hasan B.SC.(H) Physics 2230248\nTo Plot Charging and Discharging of a capacitor in RC circuit ODE with DC source by RK4 Method, Exact solution, Inbuilt solver")
plt.show()
```

5. To Plot Current in RC circuit and potential ODE with DC source by RK4 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 RK4's Method
      return ((-1)*(I))/(R*C)
   def current_exact(t):      # Current v/s time graph by ploting the solution equation of ODE
      return I0*(np.exp((-1)*t/(R*C)))
   def Vr(Vr,t):            # Voltage across resistor v/s time graph using RK4'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 RK4'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 measured:'))       # time instant
h=0.001     #step size
time_array=np.arange(0,T_fin+h,h)      # X-coordinate (time)


#   Current v/s time
I0=V/R      #current in circuit at t=0
yPointsCurrent=np.zeros(len(time_array))
yPointsCurrentExact=np.zeros(len(time_array))
yPointsCurrent[0]=I0
```

```
for i in range(len(time_array)-1):
k1=h*RC.current(yPointsCurrent[i],time_array[i])
k2=h*RC.current(yPointsCurrent[i]+(k1/2),time_array[i]+(h/2))
k3=h*RC.current(yPointsCurrent[i]+(k2/2),time_array[i]+(h/2))
k4=h*RC.current(yPointsCurrent[i]+(k3),time_array[i]+(h))
   delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)
yPointsCurrent[i+1]=yPointsCurrent[i]+delY


yPointsCurrentExact=RC.current_exact(time_array)
```

 # Solution Equation

```
solOdeintYPointsCurrent=it.odeint(RC.current,I0,time_array)
```
#odeint solution

 # Voltage across resistor v/s time

```
Vr0=V
yPointsVr=np.zeros(len(time_array))
yPointsVrExact=np.zeros(len(time_array))
yPointsVr[0]=Vr0
for i in range(len(time_array)-1):
k1=h*RC.Vr(yPointsVr[i],time_array[i])
k2=h*RC.Vr(yPointsVr[i]+(k1/2),time_array[i]+(h/2))
k3=h*RC.Vr(yPointsVr[i]+(k2/2),time_array[i]+(h/2))
k4=h*RC.Vr(yPointsVr[i]+(k3),time_array[i]+(h))
   delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)
   yPointsVr[i+1]=yPointsVr[i]+delY
yPointsVrExact=RC.VrExact(time_array)
```
 # Solution Equation

```
solOdeintYPointsVr=it.odeint(RC.Vr,Vr0,time_array)
```

#odeint solution

 # Voltage across capacitor v/s time

```
Vc0=0
yPointsVc=np.zeros(len(time_array))
yPointsVcExact=np.zeros(len(time_array))
yPointsVc[0]=Vc0
for i in range(len(time_array)-1):
k1=h*RC.Vc(yPointsVc[i],time_array[i])
k2=h*RC.Vc(yPointsVc[i]+(k1/2),time_array[i]+(h/2))
k3=h*RC.Vc(yPointsVc[i]+(k2/2),time_array[i]+(h/2))
k4=h*RC.Vc(yPointsVc[i]+(k3),time_array[i]+(h))
   delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)
   yPointsVc[i+1]=yPointsVc[i]+delY


yPointsVcExact=RC.VcExact(time_array)
```
 # Solution Equation

```
solOdeintYPointsVc=it.odeint(RC.Vc,Vc0,time_array)
```

#odeint solution

 # plot of I v/s t

```
plt.subplot(3,2,1)
plt.plot(time_array,yPointsCurrent,color='red',label="I")
plt.xlabel('Time(s)')
```

```
plt.ylabel('Current(amps)')
plt.title("Current v/s time RK4's")
plt.grid('true')
plt.legend()
plt.subplot(3,2,2)
plt.plot(time_array,yPointsCurrentExact, color='blue',label="I")
plt.xlabel('Time(s)')
plt.ylabel('Current(amps)')
plt.title("Current v/s time Solution Equation")
plt.grid('true')
plt.legend()


# plot of Vr v/s t
plt.subplot(3,2,3)
plt.plot(time_array,yPointsVr,color='red',label="Vr")
plt.plot(time_array,yPointsVc,color='blue', label="Vc")
plt.xlabel('Time(s)')
plt.ylabel('(volts)')
plt.title("Vr and Vc v/s time RK4s ")
plt.grid('true')
plt.legend()
plt.subplot(3,2,4)
plt.plot(time_array,yPointsVrExact, color='blue',label="Vr")
plt.plot(time_array,yPointsVcExact, color='red',label="Vc")
plt.xlabel('Time(s)')
plt.ylabel("(volts)")
plt.title("Vr and Vc v/s time Solution equation")
plt.grid('true')
plt.legend()


plt.subplot(3,2,5)
plt.plot(time_array,solOdeintYPointsVr, color='blue',label="Vr")
plt.plot(time_array,solOdeintYPointsVc, color='red',label="Vc")
plt.xlabel('Time(s)')
plt.ylabel("(volts)")
plt.title("Vr and Vc v/s time Odeint Solution")
plt.grid('true')
plt.legend()
plt.subplot(3,2,6)
plt.plot(time_array,solOdeintYPointsCurrent, color='blue',label="Current")
plt.xlabel('Time(s)')
plt.ylabel("(volts)")
plt.title("Current v/s time Odeint Solution equation")
plt.grid('true')
plt.legend()
plt.suptitle("Mehendi Hasan B.SC.(H) Physics 2230248\nTo Plot Current in RC circuit and potential ODE with DC source by RK4 Method, Exact solution, Inbuilt solver.")
plt.show()
```

6. To Plot Current in RL circuit ODE with DC source by RK4 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\nVariation 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)      # X-coordinate (time)


#   Current v/s time
I0=0      #current in circuit at t=0
yPointsCurrent=np.zeros(len(time_array))
yPointsCurrentExact=np.zeros(len(time_array))
yPointsCurrent[0]=I0
yPointsCurrentExact[0]=I0
```

```
for i in range(len(time_array)-1):
k1=h*diffEquation(yPointsCurrent[i],time_array[i])
k2=h*diffEquation(yPointsCurrent[i]+(k1/2),time_array[i]+(h/2))
k3=h*diffEquation(yPointsCurrent[i]+(k2/2),time_array[i]+(h/2))
k4=h*diffEquation(yPointsCurrent[i]+(k3),time_array[i]+(h))

delY=(1/6)*(k1+(2*k2)+(2*k3)+k4)
yPointsCurrent[i+1]=yPointsCurrent[i]+delY


yPointsCurrentExact=solEquation(yPointsCurrentExact,time_array)
# Solution Equation
solOdeintYPointsCurrent=it.odeint(diffEquation,I0,time_array)
#odeint solution


# plot of I v/s t
plt.subplot(1,3,1)
plt.plot(time_array,yPointsCurrent,color='red',label="I RK4")
plt.xlabel('Time(s)')
plt.ylabel("Current(Ampere)")
plt.title("Current v/s time RK4's")
plt.grid('true')
plt.legend()
plt.subplot(1,3,2)
plt.plot(time_array,yPointsCurrentExact, color='blue',label="I exact")
plt.xlabel('Time(s)')
plt.ylabel("Current(Ampere)")
plt.title("Current v/s time Solution Equation")
plt.grid('true')
plt.legend()
plt.subplot(1,3,3)
plt.plot(time_array,solOdeintYPointsCurrent, color='green',label="I ODEINT")
plt.xlabel('Time(s)')
plt.ylabel("Current(Ampere)")
plt.title("Current v/s time Odeint Solution equation")
plt.grid('true')
plt.suptitle("Mehendi Hasan B.SC.(H) Physics 2230248\nTo Plot Current in RL circuit ODE with DC source by RK4 Method, Exact solution, Inbuilt solver.")
plt.legend()
plt.show()
```