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()

1. **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()

1. **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\nRadioactive 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()

1. **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()

1. **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()

1. **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()