UNIT 6 Modiefied Euler’s Method To Solve ODE

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

# **import libraries**

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

def f(y,x): **#Differential Equation to solve**

return np.exp(x)

def f\_exact(y,x): **#Exact Equation**

return np.exp(x)

def eulersMethod(xn,yn):

return yn + h\*(f(yn,xn))

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

**#User Inputs**

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

for i in range(len(y\_values)-1):

y\_values[i+1] = y\_values[i] +(h/2)\*(f(y\_values[i],x\_values[i])+f(eulersMethod(x\_values[i],y\_values[i]),x\_values[i+1]))

y\_exact=np.array(f\_exact(y\_values,x\_values))

y\_odeint=odeint(f,y0,x\_values)

**#Ploting**

plt.subplot(3,1,1)

plt.plot(x\_values,y\_values,label="F(x) Modified Eulers",color="red")

plt.xlabel("X Points")

plt.ylabel("Y Points")

plt.grid()

plt.legend()

plt.subplot(3,1,2)

plt.plot(x\_values,y\_exact,label="F(x) Exact",color="blue")

plt.xlabel("X Points")

plt.ylabel("Y Points")

plt.grid()

plt.legend()

plt.subplot(3,1,3)

plt.plot(x\_values,y\_odeint,label="F(x) Odeint",color="green")

plt.xlabel("X Points")

plt.ylabel("Y Points")

plt.grid()

plt.legend()

plt.suptitle(" \nTo Solve First Order Differential Equation by Modified Euler Method and compare it with Exact Solution and Inbuilt Function.")

plt.show()

1. **To Plot Newton's cooling law ODE by Modified Eulers 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 f\_exact(T,t): **#Exact Equation**

Ts + ((T0-Ts)\*np.exp((-K)\*(x\_points[i+1])))

return Ts + ((T0-Ts)\*np.exp((-K)\*(t)))

def eulersMethod(xn,yn):

return yn + h\*(f(yn,xn))

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**

x\_points=np.arange(0,t+h,h)

T=np.zeros(len(x\_points))

TExact=np.zeros(len(x\_points))

T[0]=T0

TExact[0]=T0

for i in range(len(x\_points)-1):

T[i+1] = T[i] +(h/2)\*(f(T[i],x\_points[i])+f(eulersMethod(x\_points[i],T[i]),x\_points[i+1]))

TExact = f\_exact(TExact,x\_points) **#solution equation**

solOdeint=it.odeint(f,T0,x\_points) **#odeint solution**

**#ploting**

plt.subplot(3,1,1)

plt.plot(x\_points,T,label="Temperature",color="red")

plt.grid()

plt.title("Modified Eulers's Method")

plt.xlabel("Time")

plt.ylabel("Temperature of Object")

plt.legend()

plt.subplot(3,1,2)

plt.plot(x\_points,TExact,label='Temperature',color="green")

plt.grid()

plt.title("Exact Equation")

plt.xlabel("Time")

plt.legend()

plt.ylabel("Temperature of Object")

plt.subplot(3,1,3)

plt.plot(x\_points,solOdeint,label='Temperature',color="black")

plt.grid()

plt.title("Odeient Solution")

plt.xlabel("Time")

plt.ylabel("Temperature of Object")

plt.legend()

plt.suptitle(" \nTo Plot Newton's cooling law ODE by Modified Eulers method, Exact solution & Inbuilt solver")

plt.show()

1. **To Plot Radioactive Decay ODE by Modified Euler 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):

return (-1)\*(K)\*(N)

def Exact\_Equ(N,t):

return N0\*(np.exp((-1)\*(K)\*(t)))

def eulersMethod(xn,yn):

return yn + h\*(diff\_Equ(yn,xn))

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

**#taking 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 for Modified Euler method**

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

Y\_differential=np.zeros(len(t\_array)) **#Initializing array for values of dependent variable(Modified Euler's method)**

Y\_Exact=np.zeros(len(t\_array)) **# #Initializing array for values of dependent variable(Solution equation)**

Y\_differential[0] = Y\_Exact[0] = N0 **#Initial values of dependent variable Y at independent variable t=0**

for i in range(len(t\_array)-1): **#updating values of dependent variable**

#Modified Euler's Method

Y\_differential[i+1] = Y\_differential[i] +(h/2)\*(diff\_Equ(Y\_differential[i],t\_array[i])+diff\_Equ(eulersMethod(t\_array[i],Y\_differential[i]),t\_array[i+1]))

Y\_Exact=Exact\_Equ(Y\_Exact,t\_array) **# solution equation**

solOdeint=it.odeint(diff\_Equ,N0,t\_array) **# odeint solution**

**#ploting all the values of dependent variable with respect to independent variable**

plt.subplot(3,1,1)

plt.plot(t\_array,Y\_differential,color="green",label="Parent Atoms")

plt.title("Modified Euler'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='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='Parent Atoms')

plt.title("Odeint Solution")

plt.grid()

plt.xlabel("Time (Second)")

plt.ylabel("No. of parent Atoms")

plt.legend()

plt.suptitle(" \nTo Plot Radioactive Decay ODE by Modified Euler method, Exact solution & Inbuilt solver.")

plt.show()

1. **To Plot Charging and Discharging of a capacitor in RC circuit ODE with DC source by Modified Euler 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))))

def eulersMethod(xn,yn):

return yn + h\*(diff\_equ\_charging(yn,xn))

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))

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):

**#Modified Euler's Method**

Y\_diff\_charging[i+1] = Y\_diff\_charging[i] + (h/2)\*(diff\_equ\_charging(Y\_diff\_charging[i],t\_array [i])+diff\_equ\_charging(eulersMethod(t\_array[i],Y\_diff\_charging[i]),t\_array[i+1]))

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):

**#Modified Euler's Method**

Y\_diff\_discharging[i+1] = Y\_diff\_discharging[i] + (h/2)\*(diff\_equ\_discharging(Y\_diff\_discharging[i],t\_array[i])+diff\_equ\_discharging(eulersMethod(t\_array[i],Y\_diff\_discharging[i]),t\_array[i+1]))

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 all the values of dependent variable with respect to independent variable**

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("Modified Euler'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("Modified Euler'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(" \nTo Plot Charging and Discharging of a capacitor in RC circuit ODE with DC source by Modified Euler Method, Exact solution, Inbuilt solver")

plt.show()

1. **To Plot Current in RC circuit and potential ODE with DC source by Modified Euler 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 Modified Euler'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 Modified Euler'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 Modified Euler'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))))

def eulersMethod(xn,yn,f):

return yn + h\*(f(yn,xn))

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 of battery**

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)) **#initializing Y-coordinates as array of zeros of lenght time   
 array(Modified Euler method )**

yPointsCurrentExact=np.zeros(len(time\_array**)) #initializing Y-coordinates as array of zeros of lenght time   
 array(solution Equation)**

yPointsCurrent[0]=I0 **# initializing Initial value for euler's method**

for i in range(len(time\_array)-1): # **updating the array of zeros with help of euler's method and solution equation**

**#Modified Euler's Method**

yPointsCurrent[i+1] = yPointsCurrent[i] +(h/2)\*(RC.current(yPointsCurrent[i],time\_array[i])+RC.current(eulersMethod(time\_array[i],yPointsCurrent[i],RC.current),time\_array[i+1]))

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))  **#initializing Y-coordinates as array of zeros of lenght time array(Modified Euler** method )

yPointsVrExact=np.zeros(len(time\_array)) **#initializing Y-coordinates as array of zeros of lenght time array(solution** Equation)

yPointsVr[0]=Vr0 **# initializing Initial value for euler's method**

for i in range(len(time\_array)-1):  **# updating the array of zeros with help of euler's method and solution equation**

**#Modified Euler's Method**

yPointsVr[i+1] = yPointsVr[i] +(h/2)\*(RC.Vr(yPointsVr[i],time\_array[i])+RC.Vr(eulersMethod(time\_array[i],yPointsVr[i],RC.Vr),time\_array[i+1]))

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)) **#initializing Y-coordinates as array of zeros of lenght time array(Modified** Euler method )

yPointsVcExact=np.zeros(len(time\_array)) **#initializing Y-coordinates as array of zeros of lenght time array(solution** Equation)

yPointsVc[0]=Vc0 **# initializing Initial value for euler's method**

for i in range(len(time\_array)-1):  **# updating the array of zeros with help of euler's method and solution equation**

yPointsVc[i+1]=yPointsVc[i]+h\*RC.Vc(yPointsVc[i],time\_array[i]) **#Modified Euler's Method**

yPointsVc[i+1] = yPointsVc[i] +(h/2)\*(RC.Vc(yPointsVc[i],time\_array[i])+RC.Vc(eulersMethod(time\_array[i],yPointsVc[i],RC.Vc),time\_array[i+1]))

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 Modified Euler'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(3,2,4)

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 Modified Eulers ")

plt.grid('true')

plt.legend()

plt.subplot(3,2,5)

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(3,2,6)

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(3,2,3)

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(" \nTo Plot Current in RC circuit and potential ODE with DC source by Modified Euler Method, Exact solution, Inbuilt solver.")

plt.show()

1. To Plot Current in RL circuit ODE with DC source by Modified Euler 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)))

def eulersMethod(xn,yn):

return yn + h\*(diffEquation(yn,xn))

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)) **#initializing Y-coordinates as array of zeros of lenght time   
 array(Modified Euler method )**

yPointsCurrentExact=np.zeros(len(time\_array))

yPointsCurrent[0]=I0

yPointsCurrentExact[0]=I0

for i in range(len(time\_array)-1): yPointsCurrent[i+1]=yPointsCurrent[i]+h\*diffEquation(yPointsCurrent[i],time\_array[i]) **#Modified Euler's Method**

yPointsCurrent[i+1] = yPointsCurrent[i] +(h/2)\*(diffEquation(yPointsCurrent[i],time\_array[i])+diffEquation(eulersMethod(time\_array[i],yPointsCurrent[i]),time\_array[i+1]))

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 Modified Euler")

plt.xlabel('Time(s)')

plt.ylabel('Current(amps)')

plt.title("Current v/s time Modified Euler's")

plt.grid('true')

plt.legend()

plt.subplot(1,3,2)

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()

plt.subplot(1,3,3)

plt.plot(time\_array,solOdeintYPointsCurrent, color='green',label="Current")

plt.xlabel('Time(s)')

plt.ylabel("(volts)")

plt.title("Current v/s time Odeint Solution equation")

plt.grid('true')

plt.suptitle(" \nTo Plot Current in RL circuit ODE with DC source by Modified Euler Method, Exact solution, Inbuilt solver.")

plt.legend()

plt.show(