**Solution of Ordinary Differential Equations (ODE), 2nd Order-Initial Value Problem (IVP)**

**Scilab**

*//Algorithm for Eulers method for 2nd order differential equations*

clc;

clear;

clf();

function **dz**=f1(**z**)

**dz** = **z**

endfunction

function **dy**=f2(**x**, **y**)

**dy** = 2\***y** + 8\***x**\*(9 - **x**)

endfunction

x0 = 0; xn = 9; y0 = 0; z0 = 4

h = 1

n = (xn - x0)/h

x = x0:h:xn

y = zeros(1, n+1)

z = zeros(1, n+1)

y(1) = y0

z(1) = z0

for i = 1:n

y(i+1) = y(i) + h \* f1(z(i))

z(i+1) = z(i) + h \* f2(x(i), y(i))

end

disp("x = "), disp(x)

disp("y = "), disp(y)

disp("dy/dx = "), disp(z)

subplot(1,2,1)

plot(x, y, 'r-o')

xlabel("x")

ylabel("y(x)")

title("Solution y(x) using Eulers Method")

legend("y(x)")

subplot(1,2,2)

plot(x, z, 'b-s')

xlabel("x")

ylabel("dy/dx")

title("Derivative dy/dx using Eulers Method")

legend("dy/dx")

*//Algorithm for RK4 method for 2nd order differential equations*

clc;clear;clf();

function **dz**=f1(**z**)

**dz** = **z**

endfunction

function **dy**=f2(**x**, **y**)

**dy** = 2\***y** + 8\***x**\*(9 - **x**)

endfunction

x0 = 0; xn = 9; y0 = 0; z0 = 4

h = 1

n = (xn-x0)/h

x = x0:h:xn

y = zeros(1, n+1)

z = zeros(1, n+1)

y(1) = y0

z(1) = z0

for i = 1:n

k1 = h \* f1(z(i))

l1 = h \* f2(x(i), y(i))

k2 = h \* f1(z(i) + l1/2)

l2 = h \* f2(x(i) + h/2, y(i) + k1/2)

k3 = h \* f1(z(i) + l2/2)

l3 = h \* f2(x(i) + h/2, y(i) + k2/2)

k4 = h \* f1(z(i) + l3)

l4 = h \* f2(x(i) + h, y(i) + k3)

y(i+1) = y(i) + (k1 + 2\*k2 + 2\*k3 + k4)/6

z(i+1) = z(i) + (l1 + 2\*l2 + 2\*l3 + l4)/6

end

disp("x = "), disp(x)

disp("y = "), disp(y)

disp("dy/dx = "), disp(z)

subplot(1,2,1)

plot(x, y, 'r-o')

xlabel("x")

ylabel("y(x)")

title("Solution y(x) using RK-4 Method")

legend("y(x)")

subplot(1,2,2)

plot(x, z, 'b-s')

xlabel("x")

ylabel("dy/dx")

title("Derivative dy/dx using RK-4 Method")

legend("dy/dx")

#Algorithm for 2D Heat Equation Solution by Bender Schmidt Method

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

Lx, Ly = 0.1, 0.1; nx, ny = 21, 21

dx, dy = Lx/(nx-1), Ly/(ny-1)

alpha = 0.01; dt = 0.0001; nt = 1000

Fox = alpha \* dt / dx\*\*2; Foy = alpha \* dt / dy\*\*2

if Fox + Foy > 0.5:

raise ValueError("Unstable scheme! Reduce dt or refine grid.")

x = np.linspace(0, Lx, nx)

y = np.linspace(0, Ly, ny)

X, Y = np.meshgrid(x, y, indexing='ij')

T = np.exp(-5\*((X-0.5)\*\*2 + (Y-0.5)\*\*2))

T[0,:] = 0; T[-1,:] = 0

T[:,0] = 0; T[:,-1] = 0

zmax\_init = np.max(T)

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

surf = ax.plot\_surface(X, Y, T, cmap='viridis')

ax.set\_xlabel("x")

ax.set\_ylabel("y")

ax.set\_zlabel("Temperature")

ax.set\_zlim(0, zmax\_init)

ax.set\_title("2D Heat Conduction")

mappable = plt.cm.ScalarMappable(cmap='viridis')

mappable.set\_array(T)

cbar = fig.colorbar(mappable, ax=ax, shrink=0.5, aspect=10)

cbar.set\_label("Temperature")

def update(frame):

global T

Told = T.copy()

T[1:-1,1:-1] = Told[1:-1,1:-1] + \

Fox\*(Told[2:,1:-1] - 2\*Told[1:-1,1:-1] + Told[0:-2,1:-1]) + \

Foy\*(Told[1:-1,2:] - 2\*Told[1:-1,1:-1] + Told[1:-1,0:-2])

ax.clear()

surf = ax.plot\_surface(X, Y, T, cmap='viridis')

ax.set\_xlabel("x")

ax.set\_ylabel("y")

ax.set\_zlabel("Temperature")

ax.set\_zlim(0, zmax\_init)

ax.set\_title(f"2D Heat Conduction at step {frame\*20}")

return surf

ani = FuncAnimation(fig, update, frames=nt//20, interval=50, blit=False)

plt.show()