Question Paper and Solutions

1. Question 1

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import solve\_ivp

# Define the system of ODEs

def odesystem(t, y):

# y[0] represents x(t), y[1] represents y(t)

dxdt = 4 \* y[0] - y[1] + 2 \* t + 7

dydt = 2 \* y[0] + y[1] + t + 8

return [dxdt, dydt]

# Initial conditions

t0 = 0 # Initial time

y0 = [1, 2] # Initial values: x(0) = 1, y(0) = 2

# Time span for the solution

t\_span = (0, 5) # Solve from t = 0 to t = 5

t\_eval = np.linspace(t\_span[0], t\_span[1], 100) # Points where the solution is computed

# Solve the ODE system

solution = solve\_ivp(odesystem, t\_span, y0, t\_eval=t\_eval)

# Extract the solution

t = solution.t

x = solution.y[0] # x(t)

y = solution.y[1] # y(t)

# Plot the solutions

plt.figure(figsize=(10, 6))

plt.plot(t, x, 'r-', label='x(t)', linewidth=1.5)

plt.plot(t, y, 'b-', label='y(t)', linewidth=1.5)

plt.xlabel('Time t')

plt.ylabel('Solutions x(t) and y(t)')

plt.legend()

plt.title('Solution of the System of ODEs')

plt.grid()

plt.show()

# Display results

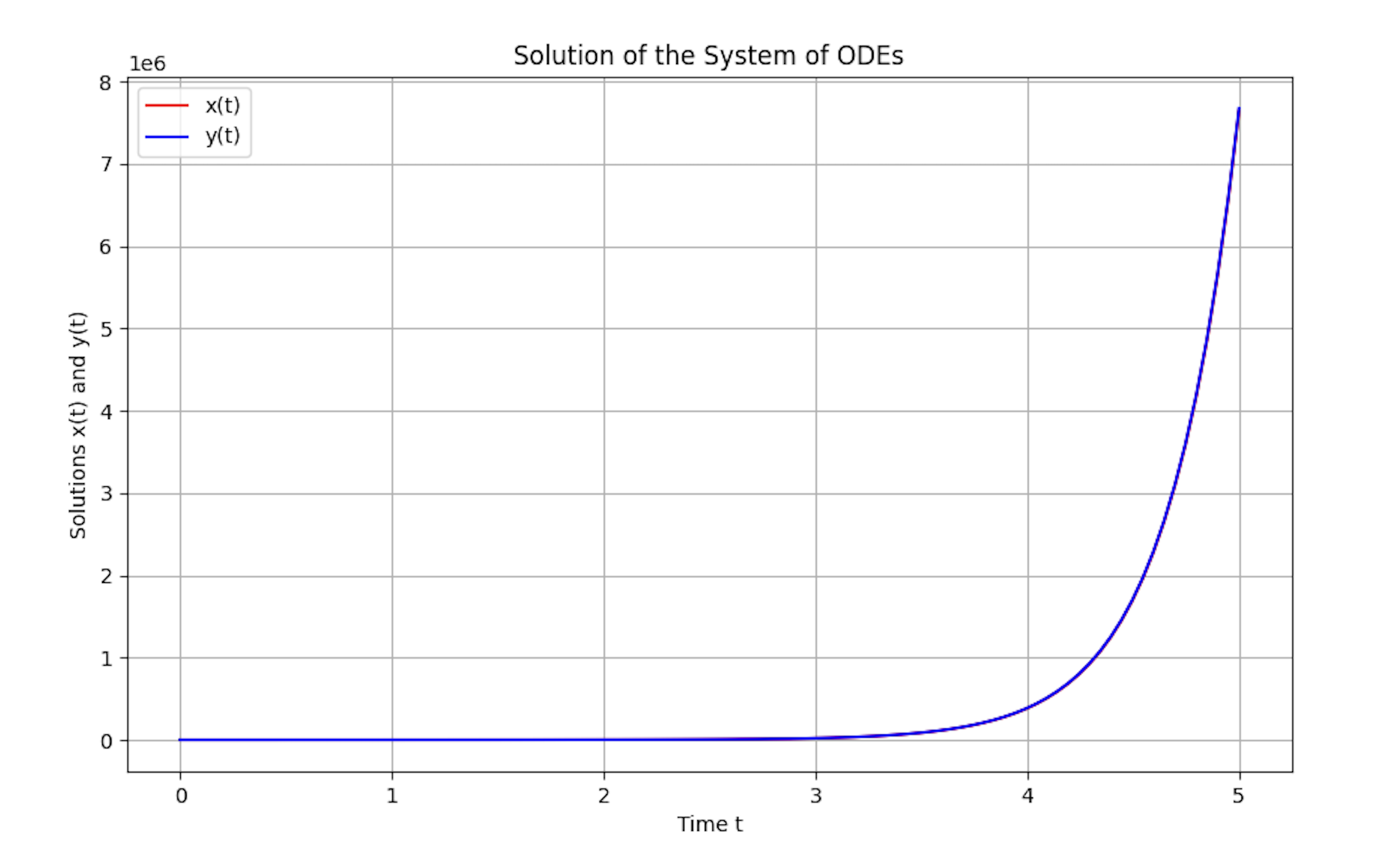
print("Time points:")

print(t)

print("Solutions [x(t), y(t)]:")

print(np.vstack((x, y)).T)

Graph



1. Question 3

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import solve\_ivp

# Define parameters

c = 1.0 # wave speed

L = 10.0 # length of the domain

N = 100 # number of spatial points

x = np.linspace(0, L, N) # spatial grid

# Initial conditions

def g(x):

return np.sin(np.pi \* x / L) # initial displacement

def v(x):

return np.zeros\_like(x) # initial velocity

# Define the second order wave equation as a system of first-order ODEs

def wave\_eq(t, u\_and\_v):

u = u\_and\_v[:N] # displacement

v = u\_and\_v[N:] # velocity

du\_dt = v

dv\_dt = c\*\*2 \* np.roll(u, -1) - 2 \* u + np.roll(u, 1) # finite difference approximation for second derivative

dv\_dt[0] = dv\_dt[-1] = 0 # boundary conditions (no flux at boundaries)

return np.concatenate([du\_dt, dv\_dt])

# Initial state: [initial displacement, initial velocity]

u0 = g(x)

v0 = v(x)

initial\_conditions = np.concatenate([u0, v0])

# Time span for the solution

t\_span = (0, 10)

t\_eval = np.linspace(0, 10, 200)

# Solve the system

sol = solve\_ivp(wave\_eq, t\_span, initial\_conditions, t\_eval=t\_eval, method='RK45')

# Plot the solution at different time points

plt.figure(figsize=(10, 6))

for i, t in enumerate([0, 2, 4, 6, 8]): # Plot at selected times

plt.plot(x, sol.y[:N, t//2], label=f't = {t}')

plt.title('Solution of the Wave Equation Over Time')

plt.xlabel('x')

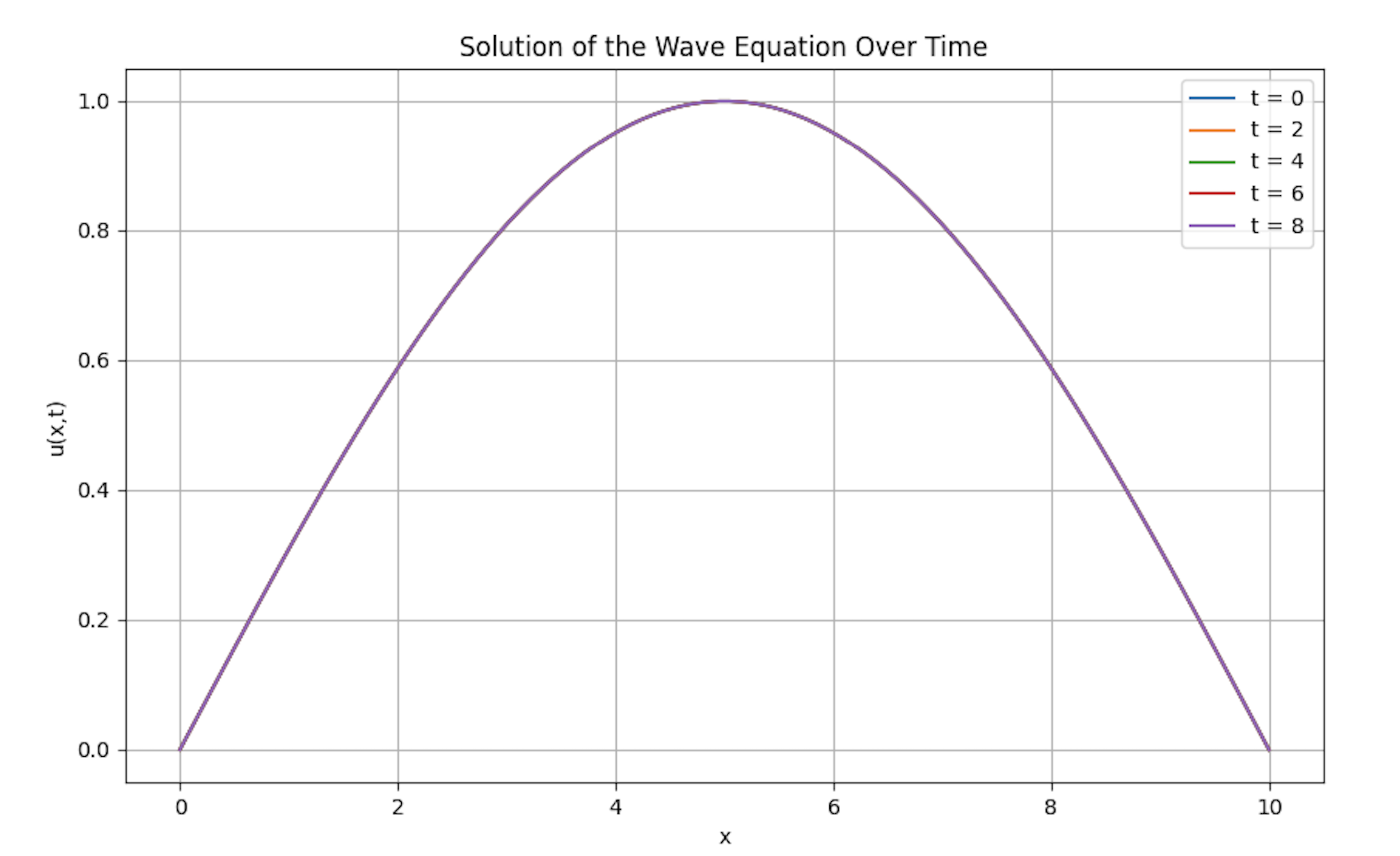
plt.ylabel('u(x,t)')

plt.legend()

plt.grid(True)

plt.show()

Graph



1. Question 4

import numpy as np

import matplotlib.pyplot as plt

# Parameters

l = 10.0 # Length of the string

T = 5.0 # Total time

Nx = 100 # Number of spatial points

Nt = 200 # Number of time steps

dx = l / (Nx - 1) # Spatial step size

dt = T / Nt # Time step size

c = 1.0 # Wave speed (assumed)

# Discretized spatial and time grids

x = np.linspace(0, l, Nx)

t = np.linspace(0, T, Nt)

# Initial conditions

g = lambda x: np.sin(np.pi \* x / l) # Initial displacement

v = lambda x: 0.0 # Initial velocity

# Initialize the solution array u(x,t)

u = np.zeros((Nx, Nt))

# Apply initial conditions at t = 0

u[:, 0] = g(x) # Initial displacement

u[:, 1] = u[:, 0] + v(x) \* dt # Initial velocity condition

# Time-stepping loop (Finite Difference Method)

for n in range(1, Nt - 1):

for i in range(1, Nx - 1):

u[i, n + 1] = 2 \* (1 - (c \* dt / dx)\*\*2) \* u[i, n] - u[i, n - 1] + (c \* dt / dx)\*\*2 \* (u[i + 1, n] + u[i - 1, n])

# Plot the solution at different times

plt.figure(figsize=(10, 6))

# Choose some time steps to plot

times\_to\_plot = [0, int(Nt / 4), int(Nt / 2), int(3 \* Nt / 4), Nt - 1]

for n in times\_to\_plot:

plt.plot(x, u[:, n], label=f't = {t[n]:.2f}s')

plt.title('Wave Equation Solution')

plt.xlabel('Position (x)')

plt.ylabel('Displacement (u)')

plt.legend()

plt.grid(True)

plt.show()

Graph

