Lab 6: Advanced Plotting and SciPy

Name: Shaheer Ziya   University Number: 3035946760

Exercise 1: Wave Function for a 2D Infinite Square Well

AIM:

The normalized wave functions for a particle in a 2D infinite square well located in the region $0 \leq x \leq L$, $0 \leq y \leq L$ are

$$\psi_{m,n}(x, y) = \frac{2}{L} \sin\left(\frac{m\pi x}{L}\right) \sin\left(\frac{n\pi y}{L}\right)$$

where (x, y) is the position of the particle, $m = 1, 2, 3, \ldots$ and $n = 1, 2, 3, \ldots$ are the quantum numbers of the state. Write a Python program that uses the `matplotlib` module to make the 3D surface plot and the 3D wireframe plot of the wave function $\psi_{4,3}(x, y)$ over the region $0 \leq x \leq L$, $0 \leq y \leq L$ side-by-side inside the same figure.

ALGORITHM:

- Prepare the values for x & y using numpy's linspace and meshgrid functions
- The range for both is $0 \leq x \leq L$, where L is set to 2 in this program.
- Obtain the Z values for the wave function setting m, n = 4, 3.
- Plot the wireframe and surface plot using the data obtained.
- Label axes and figures.

PROGRAM:

```python
# 3D Surface & Wirefram Plots
# Created by Shaheer Ziya

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np


# Constant
L, steps = 2, 500
# Parameters for Wave Fucntion
m, n = 4, 3
```

```python
def psi(x, y):
  return (2/L) * np.sin(m * np.pi * (x/L)) * np.sin(n * np.pi * (y/L))



def main():
  x = np.linspace(0, L, steps)
  y = x.copy()


  X, Y = np.meshgrid(x, y)
  Z = psi(X, Y)


  fig, axs = plt.subplots(1, 2, subplot_kw={"projection": "3d"})


  axs[0].plot_wireframe(X, Y, Z, rstride=15, cstride=15)
  axs[0].set_title("Wireframe Plot")
  axs[0].set_xlabel("x")
  axs[0].set_ylabel("y")


  axs[1].plot_surface(X, Y, Z, rstride=30, cstride=30)
  axs[1].set_title("Surface Plot")
  axs[1].set_xlabel("x")
  axs[1].set_ylabel("y")


  fig.suptitle("Wave Function for a 2D Infinite Square Well")


  plt.show()


main()
```
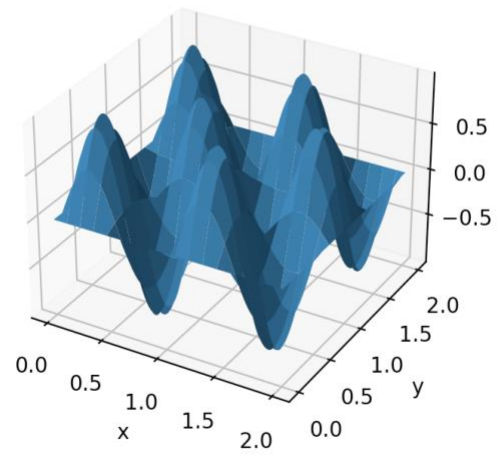
OUTPUT:

Wave Function for a 2D Infinite Square Well

Wireframe Plot
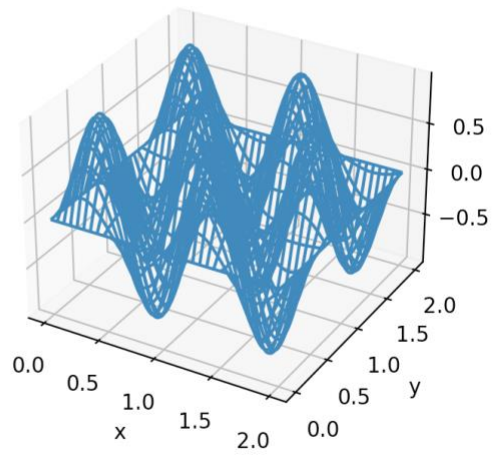
Surface Plot

Exercise 2: Mass, Center of Mass, and Moment of Inertia of a Laminar

AIM:

For a lamina occupying a region D in the x-y plane with mass density $\sigma(x, y)$, the mass M, the center of mass (x_{cm}, y_{cm}), as well as the moment of inertia about the x-axis I_x and about the y-axis I_y are given by the double integrals

$$M = \iint_D \sigma(x, y)\, dA,$$

$$x_{cm} = \frac{1}{M} \iint_D x\, \sigma(x, y)\, dA, \quad y_{cm} = \frac{1}{M} \iint_D y\, \sigma(x, y)\, dA,$$

$$I_x = \iint_D y^2\, \sigma(x, y)\, dA, \quad I_y = \iint_D x^2\, \sigma(x, y)\, dA.$$

Write a Python program that uses the `scipy.integrate` function `dblquad` to compute M, x_{cm}, y_{cm}, I_x, and I_y for a lamina occupying the region $0 \le x \le 2$, $0 \le y \le xe^{-x}$ with mass density $\sigma(x, y) = x^2 y^2$ and then outputs the results. Assume all the quantities are expressed in SI units.

ALGORITHM:

- Set up the limits of integration for the lamina along with the required density functions for integration.
- Calculate all the required properties by calling dblquad from scipy.integrate
- Print these results to the screen

PROGRAM:

```python
# Mass, Center of Mass, and Moment of Inertia of a Laminar
# Created by Shaheer Ziya


import matplotlib.pyplot as plt
import scipy.integrate as intgr
import numpy as np


# Bounds for the lamina
x = 2
y = x * np.exp(-x)


# The density function for the lamina
```

```python
def sigma(x, y):
    return (x * y) ** 2


# Functions for Centre of Mass
def xsigma(x, y):
    return x * sigma(x, y)
def ysigma(x, y):
    return y * sigma(x, y)
# Functions for Moments of Inertia
def x2sigma(x, y):
    return (x ** 2) * sigma(x, y)
def y2sigma(x, y):
    return (y ** 2) * sigma(x, y)


def main():

    M = intgr.dblquad(sigma, 0, x, 0, y)[0]
    xm = (1/M) * intgr.dblquad(xsigma, 0, x, 0, y)[0]
    ym = (1/M) * intgr.dblquad(ysigma, 0, x, 0, y)[0]
    Ix = intgr.dblquad(y2sigma, 0, x, 0, y)[0]
    Iy = intgr.dblquad(x2sigma, 0, x, 0, y)[0]


    print(f"The mass of the lamina is ~{M:.3f} kg")
    print(f"The centre of mass of the lamina is ({xm:.2f}, {ym:.2f}) (upto 2 d.p. in metres)")
    print(f"The moments of inertia of the lamina are {Ix:.5f} kgm^2 and {Iy:.5f} kgm^2")


main()
```

OUTPUT:

```
y"
The mass of the lamina is ~0.018 kg
The centre of mass of the lamina is (0.20, 1.50) (upto 2 d.p. in metres)
The moments of inertia of the lamina are ~0.04230 kgm^2 and` 0.00077 kgm^2
```

Exercise 3: Series *LRC* Circuit

AIM:

A series *LRC* circuit is composed of an inductor of inductance *L*, a resistor of resistance *R*, and a capacitor of capacitance *C* connected in series with an alternating emf $\xi(t)$. It can be shown that the charge *q* on the capacitor obeys the differential equation:

$$L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{q}{C} = \xi(t)$$

where the current in the circuit $I(t) = q'(t)$. Write a Python program to solve this equation subject to the initial conditions $q(0) = 0$ C, $I(0) = 6$ A from time $t = 0$ to 5s for the case $L = 0.5$ H, $R = 20$ Ω, $C = 0.001$ F, and $\xi(t) = 100 \sin 60t$ V by using the `scipy.integrate.odeint` method. Your program should also use the `matplotlib` module to plot the numerical solutions of $q(t)$ and $I(t)$ versus *t* as separate plots sharing the same horizontal axis.

ALGORITHM:

- Define the constants in the function
- Define the necessary relevant functions like the one to define the alternating emf
- Define the original ODE as a set simultaneous of ODEs dependent on t. (q, I)
- Initialize the initial conditions for q(0) and I(0).

$$\frac{d\vec{r}}{dt} = \left[I(t), \frac{1}{L}\left(\xi(t) - Rt - \frac{q}{C} \right) \right]^T$$

- Solve the ODEs using scipy.odeint
- Plot the resultant solution functions with the same horizontal axis on the interval they were solved on.

PROGRAM:

```
# Series LRC Circuit
# Created by Shaheer Ziya


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

```python
# L = 0.5 H, R = 20 ohms,C = 0.001 F
L, R, C = 0.5, 20, 0.001


def emf(t):
  return 100 * np.sin(60 * t)


def dr_dt(r, t: float) -> float:
  q, I = r
  dq_dt = I
  dI_dt = 1/L * (emf(t) - (R * I) - (q/C))
  return np.array([dq_dt, dI_dt])



def main():
  # r = q, I
  # Interval to solve for and plot in
  start, end = 0, 5
  # Number of partitions of the interval
  STEPS = 100
  # The array holding the partitions
  t = np.linspace(start, end, STEPS)

  # Initial conditions
  r0 = 0, 6

  # Solve the differential equation
  sol = odeint(dr_dt, r0, t)

  ### Plotting th Solution ###
  fig, axs = plt.subplots(2)
  fig.suptitle('Series LRC Circuit')
  fig.subplots_adjust(hspace=0)
  axs[0].plot(t, sol[:,0], color='blue', label='Charge q(t)')
  axs[1].plot(t, sol[:, 1], color='orange', label='Current I(t)')

  # Visual Aspects
  for i in (0,1):
```
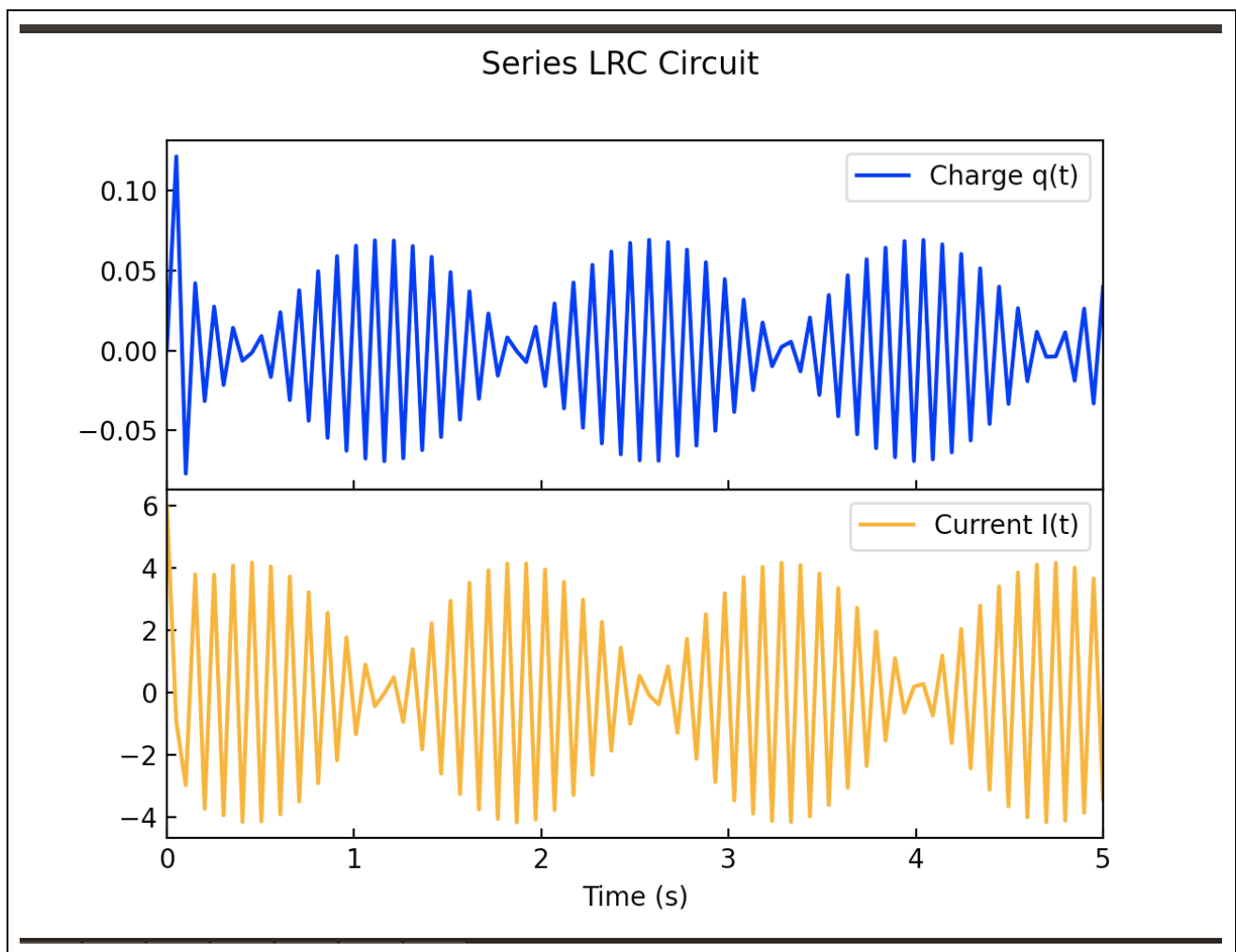
```python
    if i == 0:
        axs[i].set_xticklabels("")
    axs[i].set_xlim(start, end)
    axs[i].set_xlabel('Time (s)')
    axs[i].tick_params(direction="in")
    axs[i].legend()


    plt.show()

main()
```

OUTPUT:

Exercise 4: Legendre Polynomial

<u>AIM</u>:

Below is a table listing the data set drawn from the Legendre polynomial of degree 4, $P_4(x)$, with some noise added.

x	−1.0	−0.8	−0.6	−0.4	−0.2	0
y	0.91695	−0.19706	−0.29293	−0.04645	0.24494	0.44410
x	0.2	0.4	0.6	0.8	1.0	
y	0.31141	-0.04369	−0.42651	−0.39541	1.14994	

Write a Python program that uses the `scipy.optimize` function `curve_fit` to fit the data set to a degree-4 polynomial of x with the initial guesses of all fitting parameters set to 1, prints out the fitting parameters, as well as plots the data set, fitting result, and the polynomial $P_4(x)$ on the same graph using the `matplotlib` module and the `scipy.special` function `eval_legendre`.

<u>ALGORITHM</u>:

- Init the data
- Fit the data to a degree 4 polynomial using a user-defined function representing a degree 4 polynomial
- Create the data set for the fitted polynomial and the true legendre polynomial
- Plot the two on the same plot
- The initial fitting parameters are set to 1 by default

<u>PROGRAM</u>:

```python
# Legendre Polynomial
# Created by Shaheer Ziya

import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from scipy.special import eval_legendre

x_data = np.linspace(-1, 1, 11)
y_data = np.array([0.91695, -0.19706, -0.29293, -0.04645,
        0.24494, 0.44410, 0.31141, -0.04369, -0.42651, -0.39541, 1.14994])

def try_fit(x, a, b, c, d, e):
```

```python
    '''Degree 4 polynomial'''
    return a * (x ** 4) + b * (x ** 3) + c * (x ** 2) + d * x + e


def main():
    fitted_paramters = curve_fit(try_fit, x_data, y_data)[0]
    print(fitted_paramters)


    fig, axs = plt.subplots()



    X = np.linspace(-1, 1, 1000)


    fitted_curve = try_fit(X, *fitted_paramters)
    # axs.plot(x_data, y_data, 'o', label='Data')
    axs.plot(X, fitted_curve, '-', label='Fitted Curve')


    true_curve = eval_legendre(4, X)
    axs.plot(X, true_curve, '-', label='True Curve')


    plt.title("Legendre Polynomial $P_4(x)$")
    plt.xlabel("x")
    plt.ylabel("$y$")


    plt.legend()
    plt.show()



main()
```

OUTPUT:

Legendre Polynomial $P_4(x)$

Legendre Polynomial $P_4(x)$