Lab 6: Advanced Plotting and SciPy

Name: ____________________________ University Number: ______________

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 $0 \le x \le L$, $0 \le y \le L$ 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 $0 \le x \le L$, $0 \le y \le L$ side-by-side inside the same figure.

ALGORITHM:

1. Start
2. Import the modules to be used
3. Generate the data for the wave function $\psi_{4,3}$ versus x and y
4. Create the 3D surface plot and 3D wireframe plot of the wave function of the wave function $\psi_{4,3}$ on the same graph
5. End

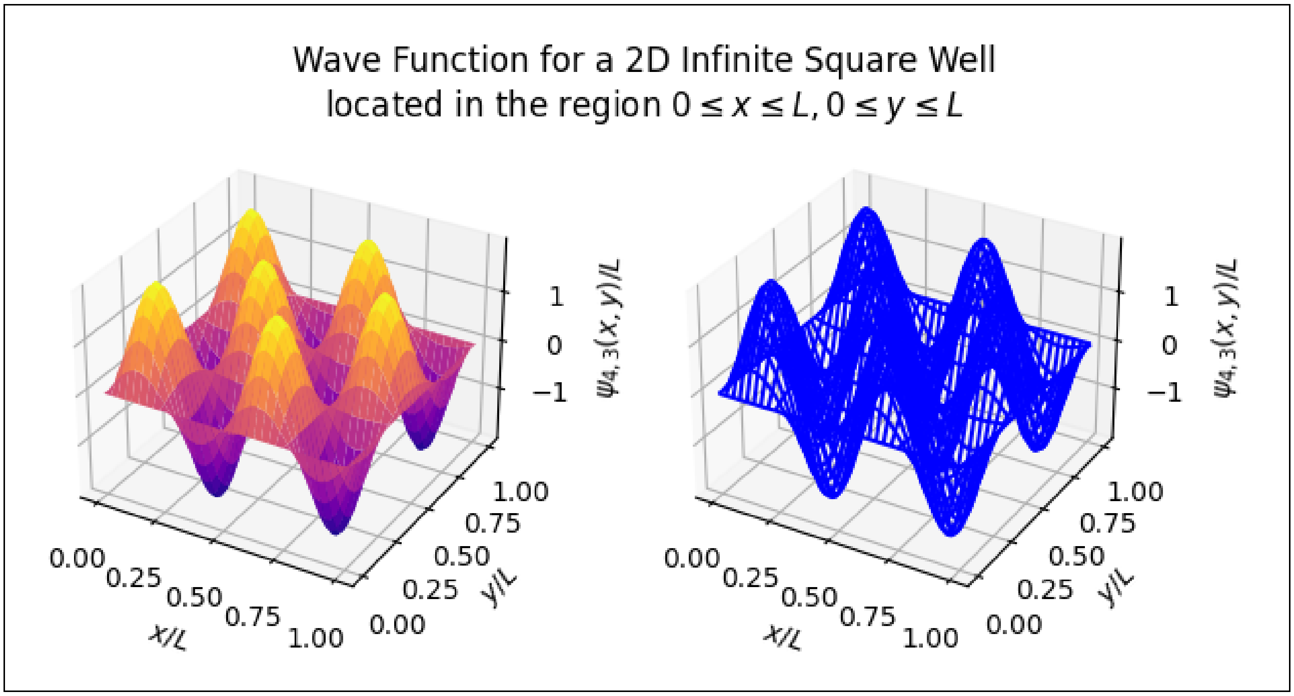
PROGRAM:

```
# Exercise 1: Wave Function for a 2D Infinite Square Well
# Written by F K Chow, HKU
# Last update: 2022/3/18

# Import the modules to be used
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

```
# Generate the data for the wave function Psi versus x and y
# Assume both x and y are expressed in units of L
m = 4
n = 3
x = np.linspace(0, 1, 101)
y = x.copy()
X, Y = np.meshgrid(x, y)
Psi = 2*np.sin(m*np.pi*X)*np.sin(n*np.pi*Y)

# Create the 3D surface plot and 3D wireframe plot of the wave function
# Psi on the same graph
fig, ax = plt.subplots(nrows=1, ncols=2,
                       subplot_kw={"projection": "3d"})
ax[0].plot_surface(X, Y, Psi, cmap=cm.plasma)
ax[1].plot_wireframe(X, Y, Psi, color="b")
for axes in ax:
    axes.set_xlabel("$x/L$")
    axes.set_ylabel("$y/L$")
    axes.set_zlabel(r"$\psi_{4, 3}(x, y)/L$")
fig.subplots_adjust(wspace=0.27, left=0.02, right=0.9)
plt.suptitle("Wave Function for a 2D Infinite Square Well\n" +\
             "located in the region " +\
             r"$0 \leq x \leq L, 0 \leq y \leq L$")
plt.show()
```

(The output is shown on the next page.)

OUTPUT:


Wave Function for a 2D Infinite Square Well
located in the region $0 \leq x \leq L, 0 \leq y \leq L$
$\psi_{4,3}(x, y)/L$
1
0
−1
x/L
y/L
0.00
0.25
0.50
0.75
1.00

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

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

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

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

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 $\sigma(x, y) = x^2y^2$ and then outputs the results. Assume all the quantities are expressed in SI units.

ALGORITHM:

1. Start
2. Import the modules to be used
3. Define the functions for the integrands
4. Define the integration limits of the integrals
5. Compute the mass, center of mass, and moment of inertia of the laminar
6. Output the results of the computation
7. End

PROGRAM:

```
# Exercise 2: Mass, Center of Mass, and Moment of Inertia of a Laminar
# Written by F K Chow, HKU
# Last update: 2022/3/18

# Import the modules to be used
import numpy as np
from scipy.integrate import dblquad
```

```
# Define the functions for the integrands
def density(y, x):
    return x**2*y**2

def xtimesdensity(y, x):
    return x*density(y, x)

def ytimesdensity(y, x):
    return y*density(y, x)

def x2timesdensity(y, x):
    return x**2*density(y, x)

def y2timesdensity(y, x):
    return y**2*density(y, x)

# Define the integration limits of the integrals
a, b = 0, 2

def hfun(x):
    return x*np.exp(-x)

def gfun(x):
    return 0

# Compute the mass, center of mass, & moment of inertia of the laminar
# Note that all the quantities are assumed to be expressed in SI units
M = dblquad(density, a, b, gfun, hfun)[0]
xcm = dblquad(xtimesdensity, a, b, gfun, hfun)[0]/M
ycm = dblquad(ytimesdensity, a, b, gfun, hfun)[0]/M
Ix = dblquad(y2timesdensity, a, b, gfun, hfun)[0]
Iy = dblquad(x2timesdensity, a, b, gfun, hfun)[0]

# Output the results of the computation
print("For a laminar occupying the region 0 <= x <= 2, "+
```

```
      "0 <= y <= xe^(-x)")
print("with mass density sigma(x, y) = x^2y^2 (all in SI units),")
print(" mass M = {:f}kg,".format(M))
print(" center of mass (xcm, ycm) = ({:f}m, {:f}m),".format(xcm, ycm))
print(" moment of inertia about the x-axis Ix = {:f}kgm^2,".format(Ix))
print(" moment of inertia about the y-axis Iy = {:f}kgm^2.".format(Iy))
```

OUTPUT:

```
For a laminar occupying the region 0 <= x <= 2, 0 <= y <= xe^(-x)
with mass density sigma(x, y) = x^2y^2 (all in SI units),
 mass M = 0.030415kg,
 center of mass (xcm, ycm) = (1.420468m, 0.248016m),
 moment of inertia about the x-axis Ix = 0.002012kgm^2,
 moment of inertia about the y-axis Iy = 0.065556kgm^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, $R = 20\ \Omega$, $C = 0.001$ 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:

1. Start
2. Import the modules to be used
3. Set up the parameters for the differential equation to be solved
4. Integrate the differential equations to obtain the numerical solution
5. Plot the numerical solution of $q(t)$ and $I(t)$ as a function of time t as separate plots on the same graph
6. End

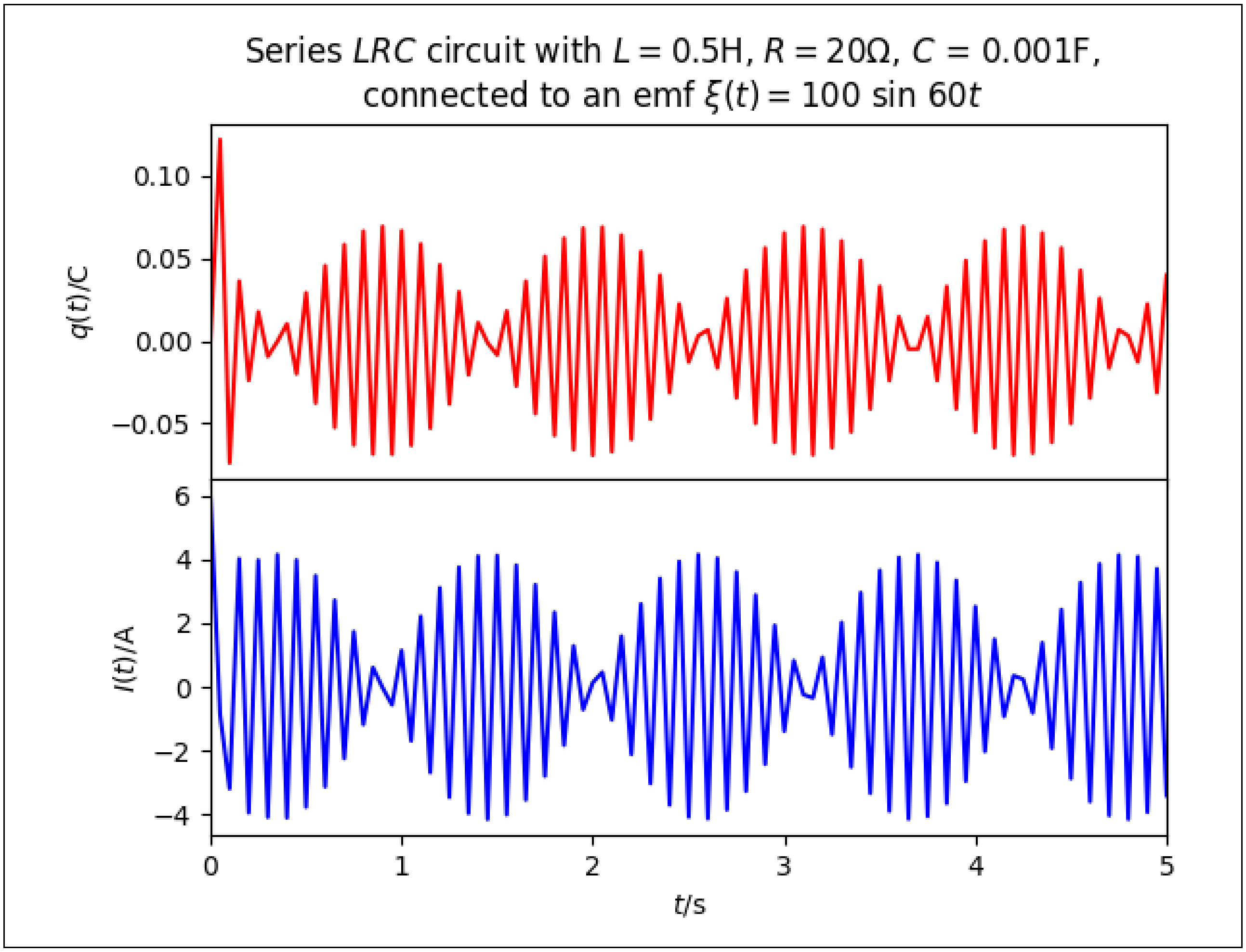
PROGRAM:

```
# Exercise 3: Series LRC Circuit
# Written by F K Chow, HKU
# Last update: 2022/3/18

# Import the modules to be used
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Set up the parameters for the DE to be solved
```

```
# All the quantities are expressed in SI units
L, R, C = 0.5, 20, 0.001
q0, I0 = 0, 6
t = np.linspace(0, 5, 101)

def drdt(r, t, L, R, C):
    """ Return dr/dt = f(r, t) at time t """
    q, I = r
    dqdt = I
    dIdt = (100/L)*np.sin(60*t) - (R/L)*I - q/(C*L)
    return dqdt, dIdt

# Integrate the DE to obtain the numerical solution
r0 = q0, I0
q, I = odeint(drdt, r0, t, args=(L, R, C)).T

# Plot the numerical solution of q(t) and I(t) versus t as separate
# plots on the same graph
fig, ax = plt.subplots(2, 1)
fig.subplots_adjust(hspace=0)
ax[0].plot(t, q, "r-")
ax[1].plot(t, I, "b-")
ax[0].set_xlim(0, 5)
ax[0].set_xticklabels("")
ax[0].set_ylabel(r"$q(t)$/C")
ax[1].set_xlim(0, 5)
ax[1].set_xlabel(r"$t$/s")
ax[1].set_ylabel(r"$I(t)$/A")
title = r"Series $LRC$ circuit with $L = ${:.1f}H, ".format(L) +\
        r"$R = {:d}\Omega$, $C$ = {:.3f}F,".format(R, C) + "\n" +\
        r"connected to an emf $\xi(t) = $100 sin 60$t$"
plt.suptitle(title)
plt.show()
```

(The output is shown on the next page.)

OUTPUT:


Series LRC circuit with $L = 0.5$H, $R = 20\Omega$, $C = 0.001$F,
connected to an emf $\xi(t) = 100 \sin 60t$
$q(t)$/C
0.10
0.05
0.00
−0.05
$I(t)$/A
6
4
2
0
−2
−4
0
1
2
3
4
5
t/s

Exercise 4: Legendre Polynomial

AIM:

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`.

ALGORITHM:

1. Start
2. Import the modules to be used
3. Compute the Legendre polynomial of degree 4, $P_4(x)$
4. Construct the data set for the fitting
5. Fit the data set to a degree-4 polynomial of x and print out the fitting parameters
6. Plot the data set, fitting result, and $P_4(x)$ on the same graph
7. End

PROGRAM:

```
# Exercise 4: Legendre Polynomial
# Written by F K Chow, HKU
# Last update: 2022/4/20

# Import the modules to be used
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from scipy.special import eval_legendre
```

```
def f(x, a, b, c, d, e):
    """ Function to calculate f(x) = ax^4 + bx^3 + cx^2 + dx + e """
    return a*x**4 + b*x**3 + c*x**2 + d*x + e

# Compute the Legendre polynomial of degree 4, P4(x)
n = 4
x = np.linspace(-1.1, 1.1, 101)
P4 = eval_legendre(n, x)

# Construct the data set for the fitting
xdata = np.linspace(-1, 1, 11)
ydata = np.array([0.91695, -0.19706, -0.29293, -0.04645, 0.24494,
                  0.44410, 0.31141, -0.04369, -0.42651, -0.39541,
                  1.14994])

# Fit the data set to a degree-4 polynomial of x and print out the
# fitting parameters
p0 = 1, 1, 1, 1, 1
popt, pcov = curve_fit(f, xdata, ydata, p0)
yfit = f(x, *popt)
print("Fitting result:")
fstr = ""
for i in range(len(popt)):
    fstr += "+ {:.6f}*x^{:d} ".format(popt[i], 4-i)
fstr = fstr.replace("+ -", "- ")
fstr = fstr.replace("x^1", "x")
fstr = fstr.replace("*x^0", "")
if fstr[:2] == "- ":
    fstr = "-" + fstr[2:]
else:
    fstr = fstr[2:]
print(" y(x) =", fstr)

# Plot the data set, fitting result, and P4(x) on the same graph
```

```
fig, ax = plt.subplots()
ax.plot(xdata, ydata, "ko", label="Data")
ax.plot(x, yfit, "r-", label="Fitting")
ax.plot(x, P4, "b--", label=r"$P_4(x)$")
ax.set_xlabel(r"$x$")
ax.set_xticks(xdata)
ax.set_xlim(-1.1, 1.1)
ax.set_ylim(-0.5, 1.5)
ax.legend()
plt.show()
```

OUTPUT:

```
Fitting result:
 y(x) = 4.598802*x^4 + 0.245836*x^3 - 4.037594*x^2 - 0.173090*x + 0.456689
```