CME 193 Project

Problem 1

Fit an ellipse to a randomly generated set of 2D data points using optimization techniques in Python.

Given a random dataset in 2D space that roughly follows an elliptical shape, your goal is to determine the parameters (a, b, r_x, r_y), where:

- (a, b) represents the center of the ellipse.
- r_x and r_y represent the radii along the x- and y-axes, respectively.

The ellipse equation we aim to fit is:

$$\frac{(x-a)^2}{r_x^2} + \frac{(y-b)^2}{r_y^2} = 1$$

Instructions

1. **Generate Data Points:**
   - Create a random dataset with M points that approximately lie on an ellipse centered at (a, b) with given radii r_x and r_y.
   - Add random noise to simulate real-world data imperfections.
   - *Hint:* Use random angles and trigonometric functions to generate points on an ellipse.
2. **Fit an Ellipse:**
   - Write a Python function `ellipseFitByDss(data)` that takes a 2-by-M array of data points and returns the best-fit ellipse parameters (a, b, r_x, r_y) by minimizing the distance of each point to the ellipse.
   - *Hint:* Use `scipy.optimize.minimize` with the "Nelder-Mead" method to find the optimal parameters.
3. **Visualize the Results:**
   - Plot the original data points.
   - Plot the fitted ellipse using the parameters from `ellipseFitByDss`.
   - *Hint:* Use the `matplotlib.pyplot` library for plotting.

Code Template

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

def generate_ellipse_data(a, b, r_x, r_y, M, noise_level=0.1):

    # Hint: Generate random angles and use cos/sin for ellipse points.
    pass

def ellipseFitByDss(data):

    # Hint: Define an objective function and use minimize with 'Nelder-Mead'.
    pass

def plot_ellipse(a, b, r_x, r_y, color='r'):

    # Hint: Use np.linspace and trigonometric functions for plotting.
    pass

# Main script
# Generate random data points around an ellipse
true_a, true_b = 5, 10     # True center
true_r_x, true_r_y = 3, 1.5  # True radii
M = 100                      # Number of points

# Generate the data
data = generate_ellipse_data(true_a, true_b, true_r_x, true_r_y, M)

# Fit the ellipse
fitted_params = ellipseFitByDss(data)

# Extract fitted parameters and plot results
fitted_a, fitted_b, fitted_r_x, fitted_r_y = fitted_params

# Plot the original data points and the fitted ellipse
plt.figure(figsize=(8, 6))
plt.scatter(data[0, :], data[1, :], label="Data Points", color="blue")
plot_ellipse(fitted_a, fitted_b, fitted_r_x, fitted_r_y, color="red")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.title("Ellipse Fitting using Downhill Simplex Search")
plt.axis('equal')
plt.show()
```

```
# Display fitted parameters
print("Fitted Ellipse Parameters:")
print(f"Center: ({fitted_a:.2f}, {fitted_b:.2f})")
print(f"Radii: (r_x = {fitted_r_x:.2f}, r_y = {fitted_r_y:.2f})")
```

Problem 2

Find a point $P = (x, y, z, w)$ in 4D space that minimizes an elastic net norm from P to five given hyperplanes.

Given a system of equations that represents five hyperplanes in 4D space, find a point P that minimizes a combination of the sum of absolute distances and the sum of squared distances to these hyperplanes. The elastic net norm we will use is defined as:

$$\alpha \sum |r_i| + (1 - \alpha) \sum r_i^2$$

where r_i are the residuals (distances from the point to each hyperplane) and α is a balancing parameter between 0 and 1.

The five equations are as follows:

$$\begin{cases} 3x - 2y + z + 4w = 1 \\ 2x + y - 3z + 2w = 3 \\ x + 2y + 2z - w = -2 \\ 4x + z + 3w = 0 \\ x - y + w = 5 \end{cases}$$

Instructions

1. **Rewrite the System in Matrix Form:**
   Express the system in matrix form $A \cdot P = b$, where:

   Hint: Write the coefficients of each equation as rows in matrix A, and the constants as entries in vector b.

2. **Define the Elastic Net Objective Function:**
   Write a Python function that calculates the elastic net objective, combining the sum of absolute distances (L1 norm) and the sum of squared distances (L2 norm) from P to each hyperplane.

   - *Hint:* Use the formula $\alpha \sum |r_i| + (1 - \alpha) \sum r_i^2$ where $r_i = A_i \cdot P - b_i$ are the residuals for each equation.
   - Use an adjustable parameter α between 0 and 1 to control the balance between the L1 and L2 components.

3. **Solve for P Using `scipy.optimize`:**
   Use `scipy.optimize.minimize` to find the point P that minimizes the elastic net objective.

   - *Hint:* Set up `minimize` to use the elastic net objective function and provide an initial guess for P (e.g., a zero vector).

4. **Compare Results Between Different α Values:**
   Experiment with different values of α (e.g., 0, 0.5, and 1) and compare the results for P and the corresponding elastic net norm.

   - *Hint:* When $\alpha = 0$, the solution should correspond to a least-squares solution. When $\alpha = 1$, it should correspond to a least absolute distances solution.
   - Plot or print out the solutions and norms for each value of α to observe the effects of the elastic net norm.

Code Template

```
import numpy as np
from scipy.optimize import minimize

# Define matrix A and vector b
A =

b =

# Elastic net objective function
def elastic_net_objective(P, A, b, alpha):
    residuals = # your code
    return #your code

# Define initial guess for P
initial_guess = np.zeros(A.shape[1])

# Minimization with scipy for different values of alpha
alphas = [0, 0.5, 1]
results = {}
for alpha in alphas:
    result = # your code
    results[alpha] = (result.x, result.fun)  # Store solution and objective value

# Display results
for alpha, (P, obj_value) in results.items():
    print(f"Alpha = {alpha}: Solution P = {P}, Elastic net norm = {obj_value}")
```

1. Question: How does the solution P vary with different values of α?

Problem 3

Given a vector-valued function $F(x, y, z)$ in three dimensions, find the roots of the system using Newton's method, $F(x, y, z) = 0$. . Specifically, you will:

- Define a function $F(x, y, z) = \begin{bmatrix} f_1(x, y, z) \\ f_2(x, y, z) \\ f_3(x, y, z) \end{bmatrix}$ where each f_i represents an equation in terms of x, y, and z.
- Calculate the Jacobian matrix $J(x, y, z)$, which is a 3×3 matrix of partial derivatives for $F(x, y, z)$.
- Implement Newton's method to find the roots of $F(x, y, z) = 0$ starting from an initial guess (x_0, y_0, z_0).
- **Function Definition**: Implement $F(x, y, z)$ and its Jacobian matrix $J(x, y, z)$.
- **Newton's Method in 3D**:
  - Use the update formula:

    $$\mathbf{x}_{n+1} = \mathbf{x}_n - J^{-1}(\mathbf{x}_n)F(\mathbf{x}_n)$$

    where $\mathbf{x}_n = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix}$ represents the current estimate.
  - Stop iterating when $||\Delta \mathbf{x}|| < \text{tol}$, or after a maximum number of iterations, `max_iter`.

Instructions

Define $F(x, y, z)$ as a vector-valued function:

$$F(x, y, z) = \begin{bmatrix} f_1(x, y, z) \\ f_2(x, y, z) \\ f_3(x, y, z) \end{bmatrix}$$

where each component $f_i(x, y, z)$ is a scalar function:

$$f_1(x, y, z) = x^2 + y^2 + z^2 - 1$$

$$f_2(x, y, z) = x + y - z - 0.5$$

$$f_3(x, y, z) = x^2 - y + z^2 - 0.25$$

Calculate the Jacobian matrix $J(x, y, z)$, where each element J_{ij} is given by the partial derivative:

$$J(x, y, z) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} \end{bmatrix}$$

Define a function `newtons_method_3d(F, J, x0, tol, max_iter)` where:

- `F` is a function that takes a vector $\mathbf{x} = [x, y, z]$ and returns the vector $F(x, y, z)$.
- `J` is a function that takes a vector $\mathbf{x} = [x, y, z]$ and returns the Jacobian matrix $J(x, y, z)$.
- `x0` is the initial guess.
- `tol` is the tolerance for stopping the iteration.
- `max_iter` is the maximum number of iterations.

Hint: Each iteration should:

1. Compute $F(x)$ and $J(x)$ for the current estimate.
2. Solve $J(x)\Delta x = F(x)$ for Δx.
3. Update $x \leftarrow x - \Delta x$.

Code Template

```
import numpy as np

# Define the vector function F and Jacobian J
def F(x):
    x, y, z = x[0], x[1], x[2]
    return # your code here

def J(x):
    x, y, z = x[0], x[1], x[2]
    return # your code here

# Newton's method for 3D
def newtons_method_3d(F, J, x0, tol=1e-7, max_iter=50):
    x = np.array(x0, dtype=float)
    # your code here
    return x

# Example usage
initial_guess = [0.5, 0.5, 0.5] # you can modify this
root = newtons_method_3d(F, J, initial_guess)
print("Root found:", root)
```

Problem 4 and 5

For Problems 4 and 5 you will analyze the Titanic dataset to explore the factors that influenced passenger survival. You will implement a classification algorithm to predict the Survived column from the other columns (besides Name).

The dataset can be found at the following URL:

http://web.stanford.edu/class/archive/cs/cs109/cs109.1166/stuff/titanic.csv

Dataset Description

The Titanic dataset contains the following columns:

- `Survived`: Indicator (1 = survived, 0 = did not survive)
- `Pclass`: Passenger class (1 = 1st class, 2 = 2nd class, 3 = 3rd class)
- `Name`: Passenger name
- `Sex`: Gender (male/female)
- `Age`: Age of the passenger
- `Siblings/Spouses Aboard`: Number of siblings or spouses aboard
- `Parents/Children Aboard`: Number of parents or children aboard
- `Fare`: Ticket fare (in pounds)

Problem 4

A: Loading and Preprocessing the Data

- Load the dataset using `pandas`.
- Convert the `Pclass` and `Sex` columns into boolean columns. Create four new columns: `Female`, `1st Class`, `2nd Class`, and `3rd Class`.
- Display the first few rows and a summary of the data. *Hint:* Convert the non-numeric columns into numeric columns (or boolean columns, for which True and False can be interpreted as 1 and 0 respectively).For example, Female would be True if the person is female, False if the person is male.

B: Creating Feature and Target Matrices

- Create a NumPy matrix `X` that includes the columns `Age`, `Siblings/Spouses Aboard`, `Parents/Children Aboard`, `Fare`, `Female`, `1st Class`, `2nd Class`, and `3rd Class`.
- Create a vector `y` from the `Survived` column.

C: Logistic Regression using Scikit-Learn

- Using `scikit-learn`, fit a logistic regression model to predict survival.
- Report the model's accuracy on the dataset and predict the survival outcome of a 30-year-old male, traveling alone, who paid 50 pounds for a 2nd-class ticket.

Problem 5

A: Defining the Model

Define a function called `probability_of_surviving(alpha, beta, X)` that computes survival probabilities for passengers:

- `alpha` is a scalar.
- `beta` is a vector of coefficients.
- `X` is an $n \times k$ matrix, where each row corresponds to a passenger and each column corresponds to a feature.

The function should return a vector with the probability that each passenger survives.

Hint: Use matrix multiplication and the sigmoid function from `scipy.special.expit`.

$$\text{probability that } x \text{ survives} = \frac{1}{1 + \exp(-(\alpha + x^T \beta))}.$$

B: Logistic Regression Loss Function

Define a function called `logistic_regression_loss(alpha_beta, X, y)` that computes the loss function:

$$L(\alpha, \beta, X, y) = \sum_{i=1}^{n} \mathrm{KL}(y_i, \hat{y}_i) + \frac{1}{2}\|\beta\|^2$$

where

- $\hat{y}_i$ is the predicted probability from `probability_of_surviving(alpha, beta, X)`.
- `KL` denotes the Kullback-Leibler divergence between true and predicted labels.

Hint: Use `scipy.special.kl_div` to calculate the KL divergence.

C: Optimizing the Model without Scikit-Learn

- Use `scipy.optimize.minimize` to find the values of α and β that minimize the logistic regression loss function.
- Compare the results of α and β to the parameters computed by `scikit-learn` in Question 3.

D: Model Prediction

Using the model from Question 6, predict the survival probability for a 30-year-old male traveling alone who paid 50 pounds for a 2nd-class ticket. Additionally, report the accuracy of your manual logistic regression model.