HW4: Least Squares Regression (100 points)

Notes:

- Question 1 needs to be typewritten.
- Questions 2, 3, 4 need to be programmed.
- Important:
    - Use proper LATEX formatting and notation for all mathematical equations, vectors, and matrices.
- For programming solution:
    - Properly add comments to your code.

A note about notation:

The notations in this homework are slightly different from the lecture notes. In lecture, we use notation for data as: (t_i, y_i) with regressor $\hat{y} = x^\top t$, x is a vector of unknown coefficients and solve $Ax = b$.
In this homework, the notation that we use for data is: (x_i, y_i) with regressor $\hat{y} = \beta^\top x$ and β is a vector of unknown coefficients to be solved.

Q1 (15 points)

Consider a dataset with m datapoints: $(x_i, y_i), i = 1, \ldots, m$. We want to minimize least squares by fitting an estimation function $\hat{y}(x) = ax^2 + bx + c$ on this data. This problem can be written in the form $A\beta = y$, such that the least squares solution for β gives the coefficients a, b, c.

What are A, β and y in terms of the variables above?

Your answer here:

A =

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ x_m^2 & x_m & 1 \end{bmatrix}$$

, beta =

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

,

y =

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Q2 (25 points)

In this problem, we would like to use a linear regressor to fit the data, where $\hat{y}(x) = ax + b$ with a, b, x being scalars. Denote $\beta_{LS} = \begin{bmatrix} a \\ b \end{bmatrix}$ to contain the regressor coefficients, and recall that the linear algebraic formula for least squares gives $\beta_{LS} = (A^\top A)^{-1} A^\top y$ with $A^\dagger = (A^\top A)^{-1} A^\top$ known as the pseudo-inverse of A.

a) Implement 3 different ways to find the regressor coefficients using the numpy package and show that they agree on the random data generated below (make sure to calcualte both a and b):

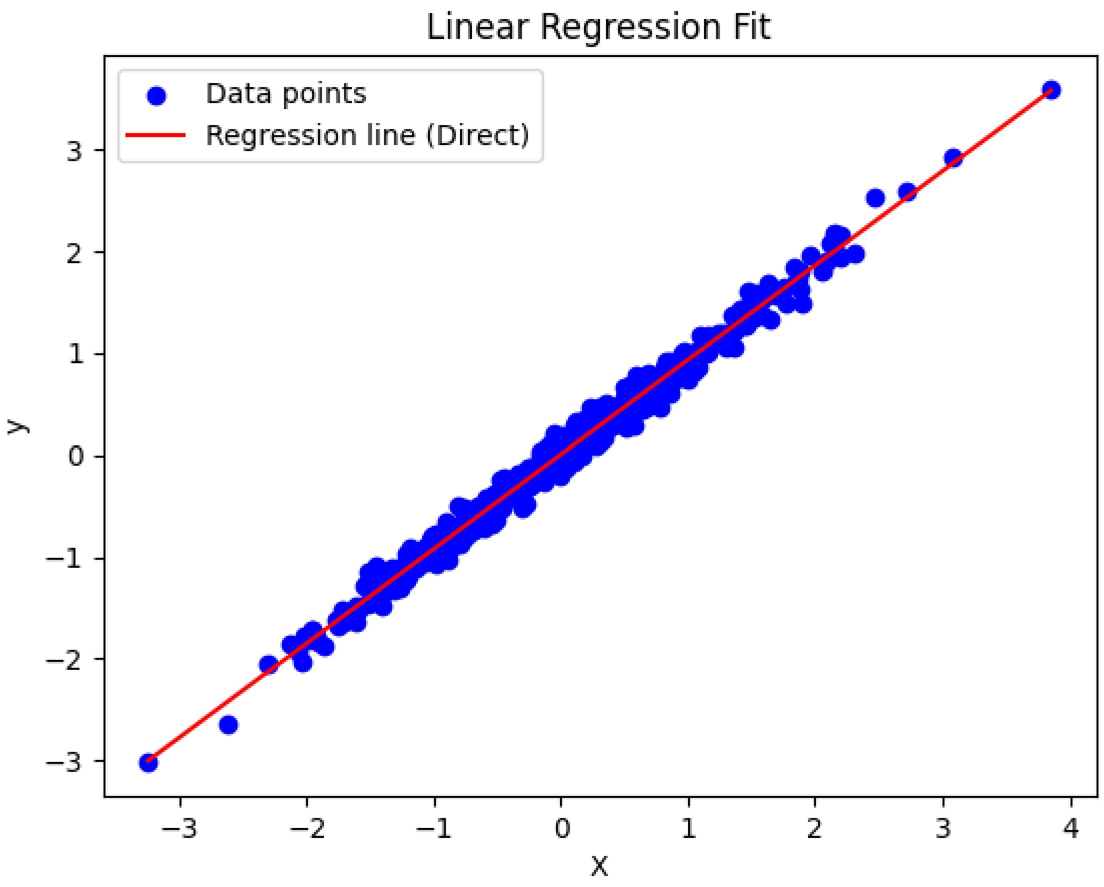
- **i.** Calculate β_{LS} directly from the with the least squares formula.
- **ii.** Use the function `np.linalg.pinv` to find the values of regressor coefficients β_{LS}.
- **iii.** Solve the problem using the builtin numpy function: `np.linalg.lstsq`

b) Plot a graph between X and y, and overlay it with the linear regression line.

```
### !!! DO NOT EDIT !!!
# starter code to generate a random least squares regression dataset with 500 points
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
from sklearn import datasets

# generate x and y
X, y =  datasets.make_regression(n_samples=500, n_features=1, n_informative=1, n_targets=
print('Shape of X is:', X.shape)
print('Shape of y is:', y.shape)
```

```
    Shape of X is: (500, 1)
    Shape of y is: (500,)
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Function to generate random data
def generate_data(n_samples=500, n_features=1, noise=0.1, random_state=42):
    np.random.seed(random_state)
    X = np.random.randn(n_samples, n_features)
    true_coef = np.random.randn(n_features, 1)
    y = X @ true_coef + noise * np.random.randn(n_samples, 1)
    return X, y, true_coef

# Generate random data
X, y, true_coef = generate_data()

# a) Three different methods to calculate the regressor coefficients

# i. Direct calculation using the least squares formula
def calculate_beta_ls_direct(X, y):
    A = np.hstack((X, np.ones((X.shape[0], 1))))
    beta_ls = np.linalg.inv(A.T @ A) @ A.T @ y
    return beta_ls.flatten()

# ii. Using np.linalg.pinv to find the values of regressor coefficients βLS
def calculate_beta_ls_pinv(X, y):
    A = np.hstack((X, np.ones((X.shape[0], 1))))
    beta_ls = np.linalg.pinv(A) @ y
    return beta_ls.flatten()

# iii. Using np.linalg.lstsq
def calculate_beta_ls_lstsq(X, y):
    A = np.hstack((X, np.ones((X.shape[0], 1))))
    beta_ls, residuals, rank, s = np.linalg.lstsq(A, y, rcond=None)
    return beta_ls.flatten()

# Calculate coefficients using all three methods
beta_ls_direct = calculate_beta_ls_direct(X, y)
beta_ls_pinv = calculate_beta_ls_pinv(X, y)
beta_ls_lstsq = calculate_beta_ls_lstsq(X, y)

# b) Plot the data and the regression line
plt.scatter(X, y, color='blue', label='Data points')
x_values = np.linspace(X.min(), X.max(), 1000)
y_values_direct = beta_ls_direct[0] * x_values + beta_ls_direct[1]
plt.plot(x_values, y_values_direct, color='red', label='Regression line (Direct)')

plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Fit')
plt.legend()
plt.show()

# Print the coefficients for comparison
print("True coefficients:", true_coef.flatten())
print("Direct method coefficients:", beta_ls_direct)
print("Pseudo-inverse method coefficients:", beta_ls_pinv)
print("Least squares function method coefficients:", beta_ls_lstsq)
```



```
True coefficients: [0.92617755]
Direct method coefficients: [0.9266581  0.00327396]
Pseudo-inverse method coefficients: [0.9266581  0.00327396]
Least squares function method coefficients: [0.9266581  0.00327396]
```

Q3 (30 points)

In this problem, we ask you to:

a) Write a function `my_func_fit (X,y)`, where `X` and `y` are column vectors of the same size containing experimental data. The function should return the values for α and β which are the scalar parameters of the estimation function $\hat{y}(x) = \alpha x^{\beta}$.

Hint: Minimize least squares in log-space, i.e. $\min \sum_i (\log(\hat{y}_i) - \log(y_i))^2$

b) Test your code on the generated sample dataset and report the coefficients. The given piece of starter code generates a logarithmic dataset.

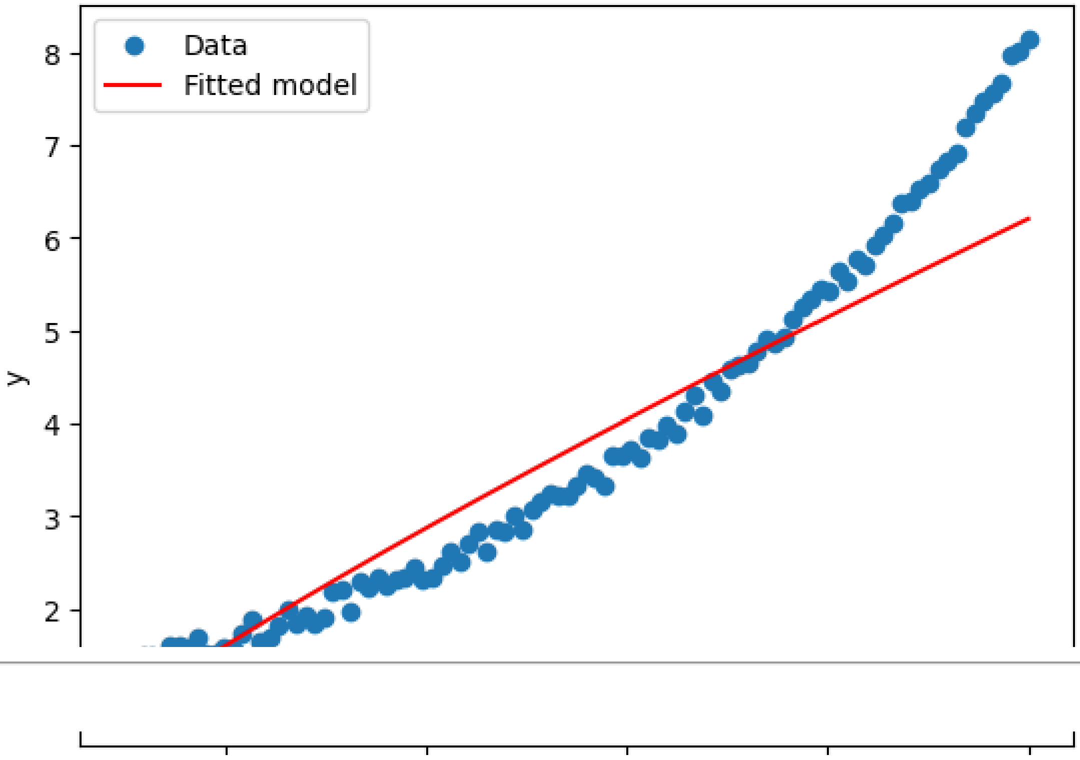
c) Plot a graph between X vs y, and overlay it with the regression line.

```
### !!! DO NOT EDIT !!!
# starter code to generate a random exponential dataset
X = np.linspace(1, 10, 101)
y = 2*(X**(0.3)) + 0.3*np.random.random(len(X))
print('Shape of X is:', X.shape)
print('Shape of y is:', y.shape)
```

```
Shape of X is: (101,)
Shape of y is: (101,)
```

```python
#######
# !!! YOUR CODE HERE !!!
# For Question 3, we need to write a function my_func_fit that fits a model to the data
# The model is y_hat = alpha * x^beta, which is a power law relationship
# We minimize the least squares in log-space

import numpy as np
import matplotlib.pyplot as plt

# Function to fit the model
def my_func_fit(X, y):
    # Take the logarithm of both sides to linearize the model: log(y) = log(alpha) + beta
    log_X = np.log(X)
    log_y = np.log(y)

    # Construct the design matrix for the linearized problem
    A = np.vstack((log_X, np.ones_like(log_X))).T

    # Solve the linear least squares problem
    beta_hat, residuals, rank, s = np.linalg.lstsq(A, log_y, rcond=None)
    alpha_hat = np.exp(beta_hat[1])
    beta = beta_hat[0]

    return alpha_hat, beta

# Starter code to generate a random exponential dataset
X = np.linspace(1, 10, 101)
y = 2**(X*0.3) + 0.3*np.random.random(len(X))  # Noise is added to the true exponential r

# Fit the model to the data
alpha_hat, beta_hat = my_func_fit(X, y)

# Report the coefficients
print(f'Estimated alpha: {alpha_hat}, Estimated beta: {beta_hat}')

# Plot the data
plt.scatter(X, y, label='Data')

# Plot the regression line
X_line = np.linspace(min(X), max(X), 1000)
y_line = alpha_hat * X_line**beta_hat
plt.plot(X_line, y_line, color='red', label='Fitted model')

plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
Estimated alpha: 0.899107040126234, Estimated beta: 0.8388293746407025
```



Q4 (30 points)

a) Write a function `my_lin_regression(f, X, y)`, where `f` is a list containing function objects to basis functions that are pre-defined, and `X` and `y` are arrays containing noisy data. Assume that `X` and `y` are the same size, i.e, $X^{(i)} \in \mathbb{R}, y^{(i)} \in \mathbb{R}$.

Return an array `beta` which represent the coefficients of the solved problem. I.e. we are solving the β which contains the coefficients in the regressor $\hat{y}(x) = \beta_0 + \beta_1 \cdot f_1(x) + \cdots + \beta_n \cdot f_n(x)$ with f_i being basis functions.

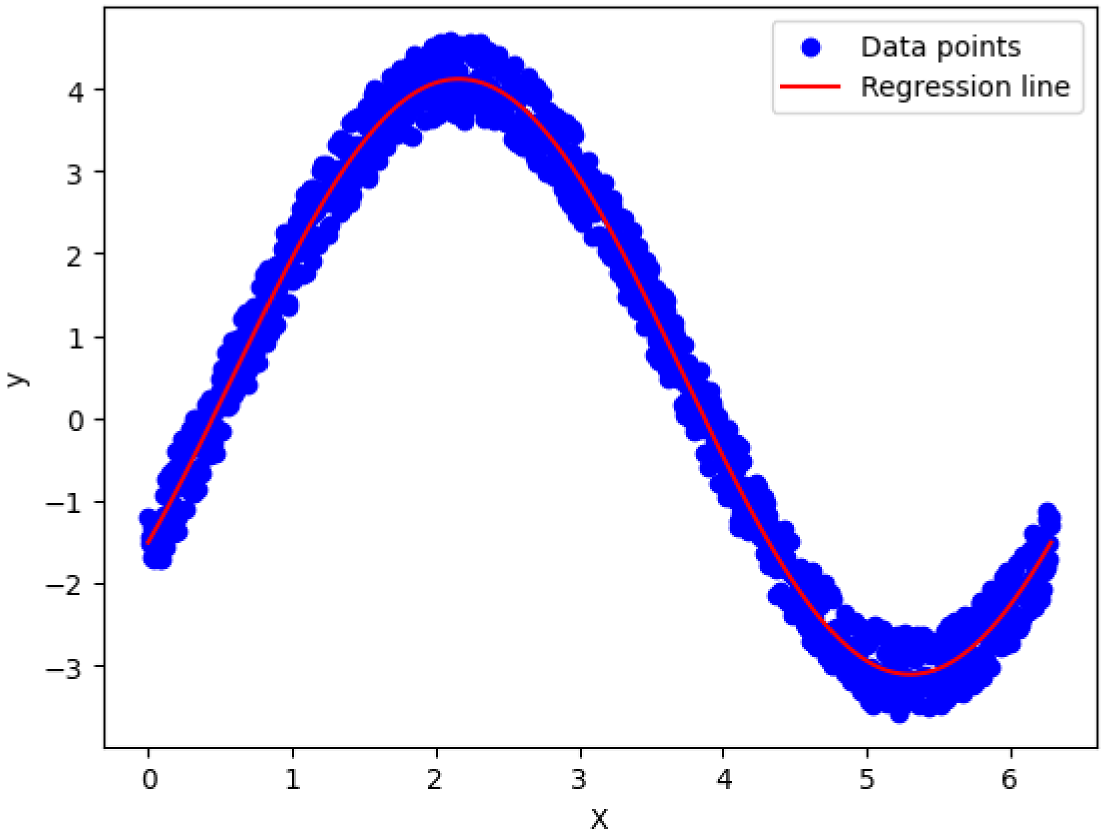
b) Also write a function `regression_plot(f,X,y,beta)` which plots a graph between `X` and `y`, and overlays it with the regression line.

Run the provided test scenarios provided below. First one uses f = [sin, cos] and second f = [exp]. Your code should plot regression lines that fit the data nicely.

```
#######
# !!! YOUR CODE HERE !!!
# Implementing the solution for Question 4 based on the provided description and test sce

# Define the linear regression function
def my_lin_regression(f, X, y):
    """
    Performs linear regression using a set of basis functions.

    Args:
    f (list): List of basis functions.
    X (numpy array): The independent variable data.
    y (numpy array): The dependent variable data.

    Returns:
    numpy array: The estimated coefficients.
    """
    # Create the design matrix by evaluating each basis function on the data
    A = np.column_stack([func(X) for func in f])

    # Add a column of ones to include the intercept term in the model
    A = np.hstack([A, np.ones((A.shape[0], 1))])

    # Use the least squares method to estimate the coefficients
    beta, _, _, _ = np.linalg.lstsq(A, y, rcond=None)
    return beta

# Define the regression plot function
def regression_plot(f, X, y, beta):
    """
    Plots the data and the regression line based on the estimated coefficients.

    Args:
    f (list): List of basis functions.
    X (numpy array): The independent variable data.
    y (numpy array): The dependent variable data.
    beta (numpy array): The estimated coefficients from the regression.
    """
    plt.scatter(X, y, label='Data points', color='blue')

    # Generate points for the regression line
    X_plot = np.linspace(np.min(X), np.max(X), 1000)
    Y_plot = sum([beta[i] * func(X_plot) for i, func in enumerate(f)]) + beta[-1]

    plt.plot(X_plot, Y_plot, label='Regression line', color='red')
    plt.xlabel('X')
    plt.ylabel('y')
    plt.legend()
    plt.show()

# Provided test scenarios

# Test 1
np.random.seed(0)  # For reproducibility
X_test1 = np.linspace(0, 2*np.pi, 1000)
```

```
y_test1 = 3*np.sin(X_test1) - 2*np.cos(X_test1) + np.random.random(len(X_test1))
f_test1 = [np.sin, np.cos]

beta_test1 = my_lin_regression(f_test1, X_test1, y_test1)
regression_plot(f_test1, X_test1, y_test1, beta_test1)

# Test 2
X_test2 = np.linspace(0, 1, 1000)
y_test2 = 2*np.exp(0.5*X_test2) + 0.25*np.random.random(len(X_test2))
f_test2 = [np.exp]

beta_test2 = my_lin_regression(f_test2, X_test2, y_test2)
regression_plot(f_test2, X_test2, y_test2, beta_test2)

(beta_test1, beta_test2)  # Return the beta coefficients for both tests

#######
```

```
### !!! DO NOT EDIT !!!
### Test-1
X = np.linspace(0, 2*np.pi, 1000)
y = 3*np.sin(X) - 2*np.cos(X) + np.random.random(len(X))
f = [np.sin, np.cos] # f1 = sin, f2 = cos

beta = my_lin_regression(f, X, y)
regression_plot(f,X,y,beta)
```



```
### !!! DO NOT EDIT !!!
### Test-2
```