Gradient descent with Linear Regression compared to the OLS matrix formulation of the beta coefficients using data from Homework 1

In [1]:
```
#defining the packages
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Defineng the linear regression function to calculate the loss
def linreg(beta, X, y):
    y_pred = np.dot(X, beta)
    loss = np.sum((y_pred - y) ** 2)
    return loss
```

In [3]:
```
#Defining  the gradient descent function for linear regression
def gradient_descent(X, y, learning_rate, num_iterations):
    n, p = X.shape
    # Initialize the beta and best_beta to 0; Initialize best_loss.
    beta = np.zeros(p)
    best_loss = float('inf')
    best_beta = np.zeros(p)

    for i in range(num_iterations):
        # Compute the gradient of the loss function at beta
        y_pred = np.dot(X, beta)
        gradient = 2 * np.dot(X.T, (y_pred - y))

        # Update beta
        beta = beta - learning_rate * gradient

        # Keep track of the best seen so far loss and parameters
        current_loss = linreg(beta, X, y)
        if current_loss < best_loss:
            best_loss = current_loss
            best_beta = beta

        # Print beta and loss update within the for loop
        if i < 10 or i > 29990:
            print(f"Iteration: {i}, Beta Values: {beta},\nBest Loss: {current_loss}")

    # Return the best beta and best loss
    return best_beta, best_loss
```

In [4]:
```
# assigning the learning rate and number of iterations
learning_rate = 0.0001
num_iterations_linear = 30000
```

In [5]:
```
# Defining the X predictor matrix and y response vector for linear regression
X = np.array([
    [1, 1, 1],
    [1, 2, 1],
    [1, 2, 2],
    [1, 3, 2],
    [1, 5, 4],
    [1, 5, 6],
    [1, 6, 5],
    [1, 7, 4],
    [1, 10, 8],
```

```
    [1, 11, 7],
    [1, 11, 9],
    [1, 12, 10]
])

y = np.array([1, 0, 1, 4, 3, 2, 5, 6, 9, 13, 15, 16])

# calling the gradient descent function for linear regression
best_beta, best_loss = gradient_descent(X, y, learning_rate, num_iterations_linear)
```

```
Iteration: 0, Beta Values: [0.015  0.1404 0.1084],
Best Loss: 539.0223544
Iteration: 1, Beta Values: [0.02657888 0.25185692 0.19406028],
Best Loss: 360.7318699583712
Iteration: 2, Beta Values: [0.03544733 0.34038133 0.26170369],
Best Loss: 248.7891117851275
Iteration: 3, Beta Values: [0.04216843 0.41073554 0.31507223],
Best Loss: 178.49778979137926
Iteration: 4, Beta Values: [0.04718834 0.46669283 0.35713079],
Best Loss: 134.35420528676465
Iteration: 5, Beta Values: [0.05086055 0.51124286 0.39022852],
Best Loss: 106.62553310358618
Iteration: 6, Beta Values: [0.05346515 0.5467544  0.41622668],
Best Loss: 89.20175472042988
Iteration: 7, Beta Values: [0.05522404 0.57510428 0.43660001],
Best Loss: 78.24715687403072
Iteration: 8, Beta Values: [0.05631306 0.59777955 0.45251696],
Best Loss: 71.35377691342129
Iteration: 9, Beta Values: [0.05687151 0.61595844 0.46490333],
Best Loss: 67.00995746756539
Iteration: 29991, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.10088344257624
Iteration: 29992, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.10088344257625
Iteration: 29993, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.10088344257624
Iteration: 29994, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.100883442576254
Iteration: 29995, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.10088344257624
Iteration: 29996, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.10088344257626
Iteration: 29997, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.100883442576226
Iteration: 29998, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.10088344257625
Iteration: 29999, Beta Values: [-2.26303789  1.54972927 -0.2385295 ],
Best Loss: 34.10088344257623
```

In [6]:

```
# Print the best beta and best loss from linear regression
print("Best Beta Values:", best_beta)
print("Best Loss:", best_loss)
```

```
Best Beta Values: [-2.26303788  1.54972927 -0.2385295 ]
Best Loss: 34.1008834425762
```

In [7]:

```
# Calculate betas using "OLS Matrix Formulation"
Transpose_X = np.transpose(X)
Transpose_X_X = np.dot(Transpose_X, X)
Transpose_X_X_INV = np.linalg.inv(Transpose_X_X)
Transpose_X_Y = np.dot(Transpose_X, y)
betas_matrix_calc = np.dot(Transpose_X_X_INV, Transpose_X_Y)

# Printing the calculated betas using the OLS Matrix Formulation
print("Betas calculated using OLS Matrix Formulation (Linear Regression):")
print(betas_matrix_calc)
```

```
Betas calculated using OLS Matrix Formulation (Linear Regression):
[-2.2630379   1.54972927 -0.2385295 ]
```

Comparing the best beta values obtained from Homework 1, which are

Beta 0: -2.263037902536326 Beta 1: 1.5497292675976115
Beta 2: -0.2385294955827928 In comparison to the best beta values(in homework1) from the gradient decent, there is no drastic difference in the values

The above python program shows the gradient desent for linear regression we are giving the four parameter with varible X as input data y as targeted values, learning_rate (step size for updating the weights), and num_iterations (number of iterations to run) The program initiates by initializing the weight vector (w) and other variables necessary for tracking the best weights and the lowest loss. It proceeds by entering a loop, iteratively adjusting the weights using the gradient of the mean squared error loss function.Within each iteration, the program calculates predicted values (y_pred), computes the gradient vector, updates the weights with respect to the learning rate, and computes the current loss. It diligently maintains records of the best weights and the lowest loss encountered throughout the process. Ultimately, upon completing the specified number of iterations, the program returns the best weights and their corresponding minimal loss. The primary objective of this program is to determine the weights that minimize the mean squared error in a linear regression model

This is an additional code of pseudocode of shiny serve which shows the plot of Deterministic Gradient Descent Optimization

In [8]:

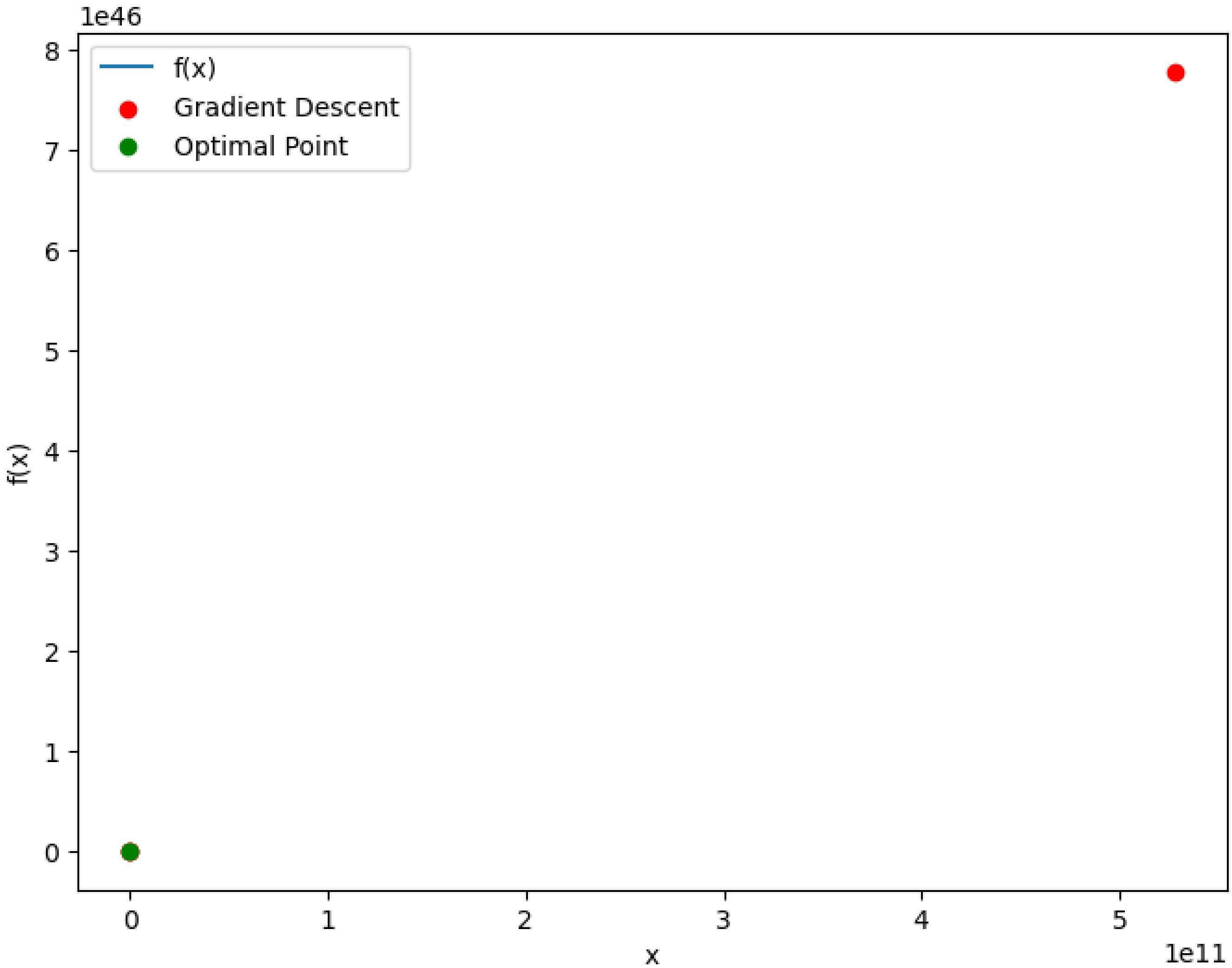
```
# Define the function f(x)
def f(x):
    return x**4 - 10*x**2 + 2 - x
```

In [9]:

```
# Initialize empty lists
x_values=[]
fx_values=[]

alpha = float(input("Enter the learning rate: "))
x = float(input("Enter the initial condition (x0): "))
num_iterations = int(input("Enter the number of iterations: "))


# Perform gradient descent for the function f(x)
for i in range(1, num_iterations + 1):
    gradient = 4 * x**3 - 20 * x - 1
    x = x - alpha * gradient
    x_values.append(x)
    fx_values.append(f(x))

min_fx = min(fx_values)
optimal_x = x_values[fx_values.index(min_fx)]

print(f"Optimal x for f(x): {optimal_x}")
print(f"Minimum f(x): {min_fx}")

x_range = np.linspace(-3, 3, 400)
```

```python
plt.figure(figsize=(8, 6))
plt.plot(x_range, f(x_range), label="f(x)")
plt.scatter(x_values, fx_values, color='red', label="Gradient Descent")
plt.scatter(optimal_x, min_fx, color='green', marker='o', label="Optimal Point")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.show()
```

```
Enter the learning rate: 1
Enter the initial condition (x0): 2
Enter the number of iterations: 3
Optimal x for f(x): 11.0
Minimum f(x): 13422.0
```



In [ ]:
```
!jupyter nbconvert --to webpdf --allow-chromium-download assignment3.ipynb
```

In [ ]:
```
jupyter nbconvert --to webpdf --allow-chromium-download assignment3.ipynb
```