1.1)

k	weight (x_k)	mpg (y_k)
1	2.5	30
2	3.0	25
3	3.5	22
4	4.0	18
5	4.5	15

e_k

$e_1 = y - \hat{y} = y - (-5(2.5) + 40) = y - 27.5 = 2.5)^2 = 6.25$

$e_2 = y - \hat{y} = y - (-5(3) + 40) = y - 25 = 0)^2 = 0$

$e_3 = y - \hat{y} = y - (-5(3.5) + 40) = y - 22.5 = -0.5)^2 = .25$

$e_4 = y - \hat{y} = y - (-5(4) + 40) = y - 20 = -2)^2 = 4$

$e_5 = y - \hat{y} = y - (-5(4.5) + 40) = y - 17.5 = -2.5)^2 = 6.25$

~~Sum = 22.5~~

~~$L = 22.5 / 5$ Sum 4.25 Loss~~

~~model $\hat{p} = mx_i$~~

$\hat{y}_k = m x_k + b$

$m = -5$, $b = 40$

Sum = 16.75

Loss = 16.75/5 = 3.35

1.2)

$z = y^2 + x^2$

$y = 2t$

$x = t^2$

$\frac{\partial z}{\partial x} = 2x$

$\frac{\partial z}{\partial y} = 2y$

$\frac{\partial x}{\partial t} = 2t$

$\frac{\partial y}{\partial t} = 2$

$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial x}\left(\frac{\partial x}{\partial t}\right) + \frac{\partial z}{\partial y}\left(\frac{\partial y}{\partial t}\right)$

$\Rightarrow \frac{\partial z}{\partial t} = 2x(2t) + 2y(2)$

$= 4xt + 4y$

$L = L_1 + L_2 + L_3 + L_4 + L_5 \Rightarrow L_k = \frac{1}{n} e_k^2$

$\Rightarrow \sum_k \frac{1}{n} e_k^2$

1.3) $\hat{y}_k = m x_k + b \quad e_k = (y_k - \hat{y}_k) \quad L_k = \left(\frac{1}{n}\right) e_k^2$

A)

$\frac{\partial L_k}{\partial e_k} = \frac{2}{n} e_k \;\Big|\; \frac{\partial e_k}{\partial \hat{y}_k} = -1 \;\Big|\; \frac{\partial \hat{y}_k}{\partial m} = x_k \Rightarrow \frac{\partial L}{\partial m} = \left(\left(\frac{2}{5}\right) e_k\right) \cdot (-1)(x_k) = -\frac{2}{5} e_k x_k$

$k = 1 \quad x_1 = 2.5 \quad y_1 = 30 \quad m = -5 \quad b = 40 \quad e_1 = 30 - \left((-5)(2.5) + 40\right) = 2.5$

$\frac{\partial L}{\partial m} = -\frac{2}{5}(2.5)(2.5) = -2.5$

B)

$\frac{\partial L_k}{\partial b} = \underbrace{\left(\frac{\partial L_k}{\partial e_k}\right)}_{\frac{2}{5} e_k} \underbrace{\left(\frac{\partial e_k}{\partial \hat{y}_k}\right)}_{-1} \underbrace{\left(\frac{\partial \hat{y}_k}{\partial b}\right)}_{1} = \left(\frac{2}{5} e_k\right)(-1)(1) = -\frac{2}{5}(2.5) = -1$

1.4)

$$\frac{\partial L_k}{\partial m} = -\frac{2}{n}(y_k - \hat{y}_k)x_k$$

$$\frac{\partial L_k}{\partial b} = -2/n(y_k - \hat{y}_k)$$

$$\frac{\partial L}{\partial m} = \frac{\partial L_1}{\partial m} + \frac{\partial L_2}{\partial m} + \frac{\partial L_3}{\partial m} + \frac{\partial L_4}{\partial m} + \frac{\partial L_5}{\partial m}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L_1}{\partial b} + \frac{\partial L_2}{\partial b} + \frac{\partial L_3}{\partial b} + \frac{\partial L_4}{\partial b} + \frac{\partial L_5}{\partial b}$$

k	x_k	y_k	$\hat{y}$	$y_k - \hat{y}_k$	$\frac{\partial L_k}{\partial m}$	$\frac{\partial L_k}{\partial b}$
1	2.5	30	27.5	2.5	$-\frac{2}{5}(2.5)(2.5) = -2.5$	$-\frac{2}{5}(2.5) = -1$
2	3.0	25	25	0	0	0
3	3.5	22	22.5	-0.5	$-\frac{2}{5}(-.5)(3.5) = .7$	$-\frac{2}{5}(-.5) = 2$
4	4.0	18	20	-2	$-\frac{2}{5}(-2)(4) = 3.2$	$-\frac{2}{5}(-2) = .8$
5	4.5	15	17.5	-2.5	$-\frac{2}{5}(-2.5)(4.5) = 4.5$	$-\frac{2}{5}(-2.5) = 1$
					Sum = 5.9	Sum = 1

$\frac{\partial L}{\partial m} = 5.9$ $\quad \frac{\partial L}{\partial b} = 1.0$

to Reduce loss we shrink both m & b

$$m_{new} = m - \alpha \frac{\partial L}{\partial m} = -5 - .1(5.9) = \boxed{-5.59}$$

$$b_{new} = b - \alpha \frac{\partial L}{\partial b} = \boxed{39.9}$$

Homework 3

Linear Regression with Gradient Descent (from scratch)

In [1]:
```
import numpy as np  # Only for data loading and plotting
import matplotlib.pyplot as plt

class LinearRegressionGD:
    """
    Linear Regression using Gradient Descent
    Model: y_hat = m * x + b

    This implementation follows the mathematical framework from Part 1:
    - Compute gradient contribution from each data point
    - Sum all contributions to get total gradient
    - Update parameters
    """

    def __init__(self, learning_rate=0.01, num_iterations=1000):
        """
        learning_rate: step size for gradient descent (α)
        num_iterations: number of iterations to run
        """
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.m = 0  # slope
        self.b = 0  # intercept
        self.loss_history = []  # track loss over iterations

    def predict(self, x):
        """
        Make a prediction for a single input value

        Input:
            x: a single input value (scalar)
        Returns:
            y_hat: predicted value (scalar)
        """
        return self.m * x + self.b

    def optimize(self, X, y):
        """
        Train the model using gradient descent

        Inputs:
            X: list or array of input values
            y: list or array of true output values
        Returns:
            self (for method chaining)
        """
        n = len(X)
        self.loss_history = []

        for iteration in range(self.num_iterations):
            grad_m = 0.0
            grad_b = 0.0
            total_loss = 0.0

            # Loop over each data point individually (no vectorized gradients)
            for x_i, y_i in zip(X, y):
                y_hat = self.predict(x_i)
```

```
                error = y_hat - y_i

                grad_m += (2.0 / n) * error * x_i
                grad_b += (2.0 / n) * error
                total_loss += error ** 2

            self.m -= self.learning_rate * grad_m
            self.b -= self.learning_rate * grad_b

            self.loss_history.append(total_loss / n)

        return self
```

In [2]:

```
# Data from Problem 1.1
X_small = [2.5, 3.0, 3.5, 4.0, 4.5]
y_small = [30, 25, 22, 18, 15]

# Train model
model = LinearRegressionGD(learning_rate=0.01, num_iterations=100)
model.optimize(X_small, y_small)

print(f"\nFinal parameters:")
print(f"m (slope): {model.m:.4f}")
print(f"b (intercept): {model.b:.4f}")
print(f"Final Loss: {model.loss_history[-1]:.4f}")
```

```
Final parameters:
m (slope): 4.4800
b (intercept): 4.7464
Final Loss: 73.3700
```

In [3]:

```
# Verification: 1 iteration from m=-5, b=40
verify_model = LinearRegressionGD(learning_rate=0.01, num_iterations=1)
verify_model.m = -5
verify_model.b = 40
verify_model.optimize(X_small, y_small)

print("After 1 iteration with lr=0.01 from m=-5, b=40:")
print(f"m: {verify_model.m:.4f}")
print(f"b: {verify_model.b:.4f}")
print("Compare these values to your hand calculation from Problem 1.4 Part B Q3.")
```

```
After 1 iteration with lr=0.01 from m=-5, b=40:
m: -5.0590
b: 39.9900
Compare these values to your hand calculation from Problem 1.4 Part B Q3.
```

In [4]:

```
from urllib.request import urlretrieve
import pandas as pd

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.da
urlretrieve(url, "auto-mpg.data")

# Only the numeric columns (ignore car_name because it has spaces)
column_names_num = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
                    'acceleration', 'model_year', 'origin']

data = pd.read_csv(
    "auto-mpg.data",
    sep=r"\s+",
    header=None,
    names=column_names_num,
    usecols=list(range(8)),   # read only the first 8 columns
    na_values="?",
    engine="python",
)
```

```
data = data.dropna()

X_full = (data["weight"].astype(float).to_numpy() / 1000)
y_full = data["mpg"].astype(float).to_numpy()

print(f"Dataset size: {len(X_full)} vehicles")
```

```
Dataset size: 392 vehicles
```

In [5]:

```
# Train model on full dataset
# Start with the same learning rate as before and adjust if needed
model_full = LinearRegressionGD(learning_rate=0.01, num_iterations=1000)
model_full.optimize(X_full, y_full)

print(f"\nFull dataset - Final parameters:")
print(f"m (slope): {model_full.m:.4f}")
print(f"b (intercept): {model_full.b:.4f}")
print(f"Final Loss: {model_full.loss_history[-1]:.4f}")

x_test = 3.0
print(f"Predicted MPG for 3000 lbs (x=3.0): {model_full.predict(x_test):.4f}")
```

```
Full dataset - Final parameters:
m (slope): -4.1255
b (intercept): 34.9596
Final Loss: 28.2222
Predicted MPG for 3000 lbs (x=3.0): 22.5832
```

Problem 2.2 Questions (fill in after running)

- Slope interpretation: For every 1000 lb increase in vehicle weight, MPG changes by approximately `m` MPG.
- Relationship sense-check: Heavier cars usually get lower MPG, so a negative slope is expected.
- Learning rate note: Keep `0.01` if loss decreases smoothly; reduce it if the loss oscillates/diverges.

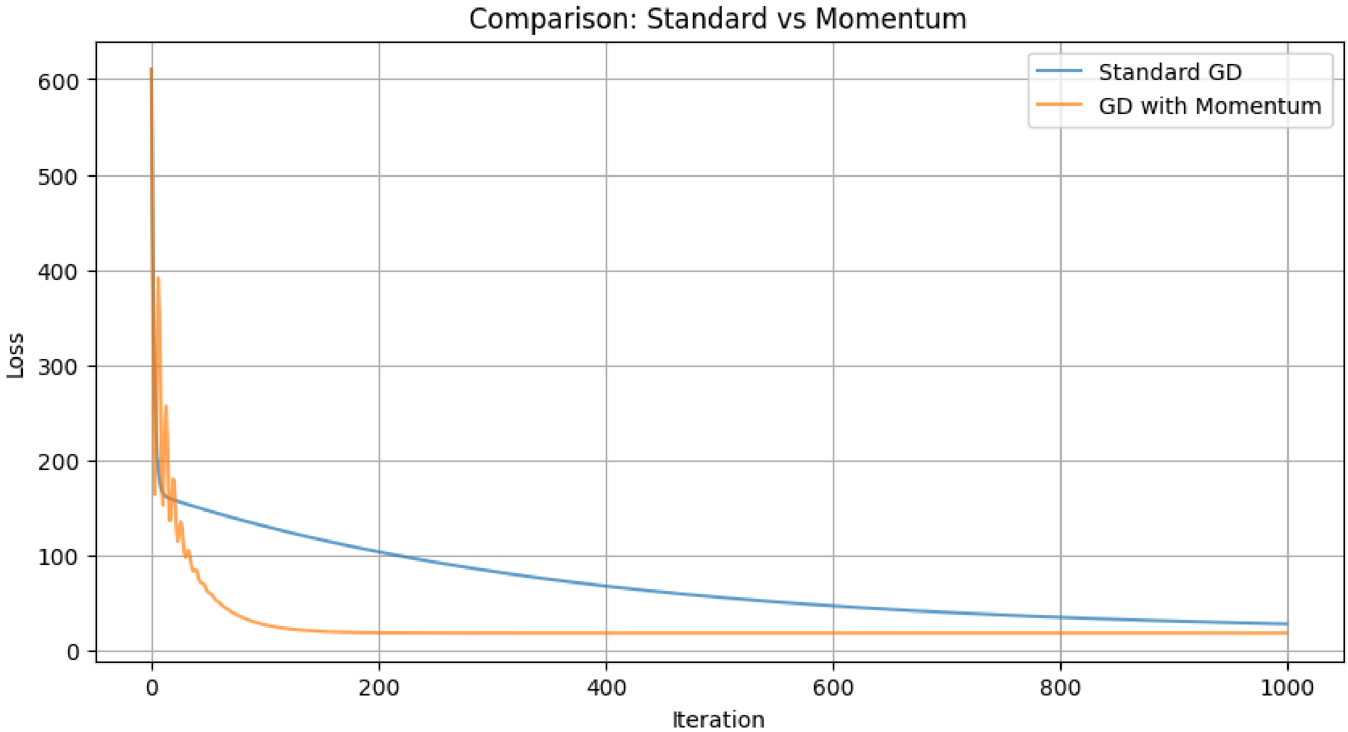
In [6]:

```
class LinearRegressionGDMomentum:
    """
    Linear Regression using Gradient Descent with Momentum

    Momentum update rule (from Going Beyond 1.4):
        v_m = β * v_m + gradient_m
        m = m - α * v_m
    """

    def __init__(self, learning_rate=0.01, num_iterations=1000, momentum=0.9):
        """
        learning_rate: step size α
        num_iterations: number of iterations
        momentum: momentum coefficient β (typically 0.9)
        """
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.momentum = momentum
        self.m = 0
        self.b = 0
        self.v_m = 0  # velocity for m
        self.v_b = 0  # velocity for b
        self.loss_history = []

    def predict(self, x):
        """Same as before"""
        return self.m * x + self.b
```

```
    def optimize(self, X, y):
        """
        Train using gradient descent with momentum

        Inputs:
            X: list or array of input values
            y: list or array of true output values
        Returns:
            self
        """
        n = len(X)
        self.loss_history = []

        for _ in range(self.num_iterations):
            grad_m = 0.0
            grad_b = 0.0
            total_loss = 0.0

            for x_i, y_i in zip(X, y):
                y_hat = self.predict(x_i)
                error = y_hat - y_i

                grad_m += (2.0 / n) * error * x_i
                grad_b += (2.0 / n) * error
                total_loss += error ** 2

            self.v_m = self.momentum * self.v_m + grad_m
            self.v_b = self.momentum * self.v_b + grad_b

            self.m -= self.learning_rate * self.v_m
            self.b -= self.learning_rate * self.v_b

            self.loss_history.append(total_loss / n)

        return self
```

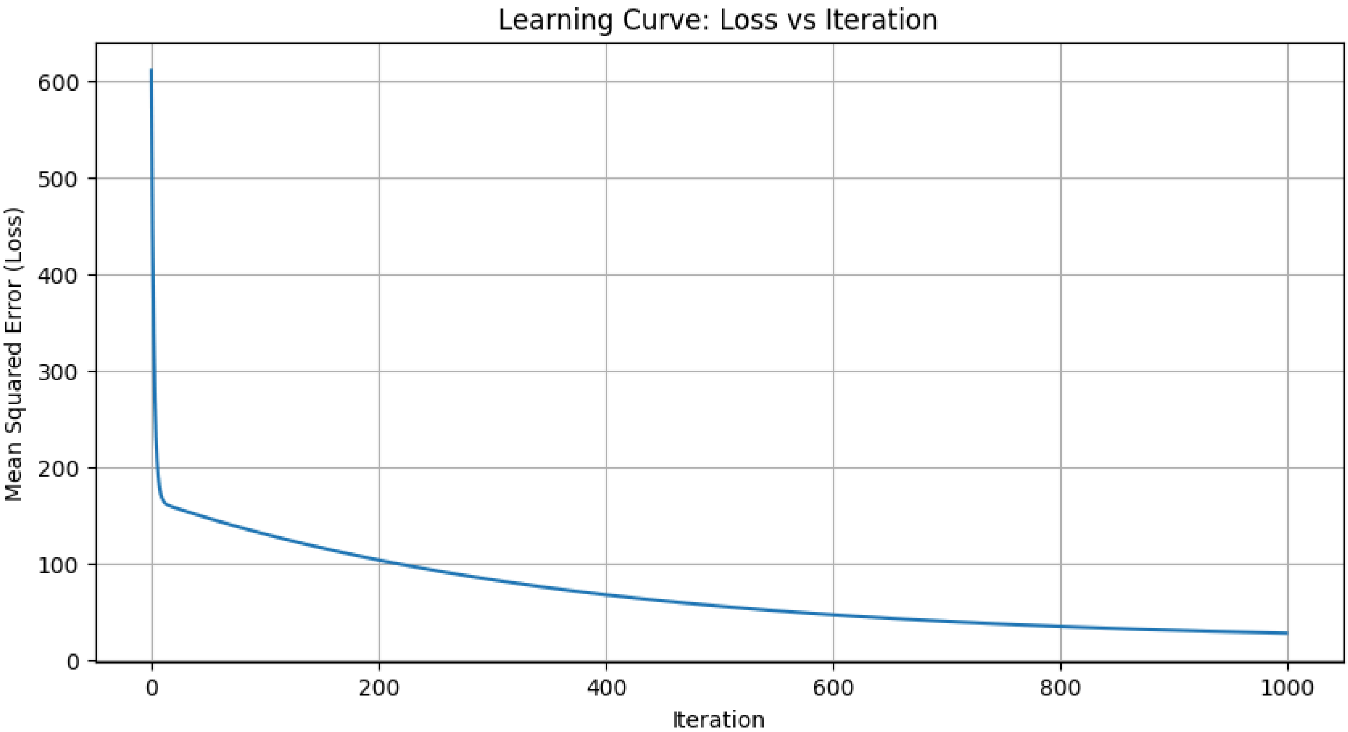
In [7]:

```
# Standard gradient descent
model_standard = LinearRegressionGD(learning_rate=0.01, num_iterations=1000)
model_standard.optimize(X_full, y_full)

# Gradient descent with momentum
model_momentum = LinearRegressionGDMomentum(learning_rate=0.01, num_iterations=1000,
model_momentum.optimize(X_full, y_full)

# Plot comparison
plt.figure(figsize=(10, 5))
plt.plot(model_standard.loss_history, label='Standard GD', alpha=0.7)
plt.plot(model_momentum.loss_history, label='GD with Momentum', alpha=0.7)
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Comparison: Standard vs Momentum')
plt.legend()
plt.grid(True)
plt.show()

print(f"Standard GD final loss: {model_standard.loss_history[-1]:.4f}")
print(f"Momentum GD final loss: {model_momentum.loss_history[-1]:.4f}")
```



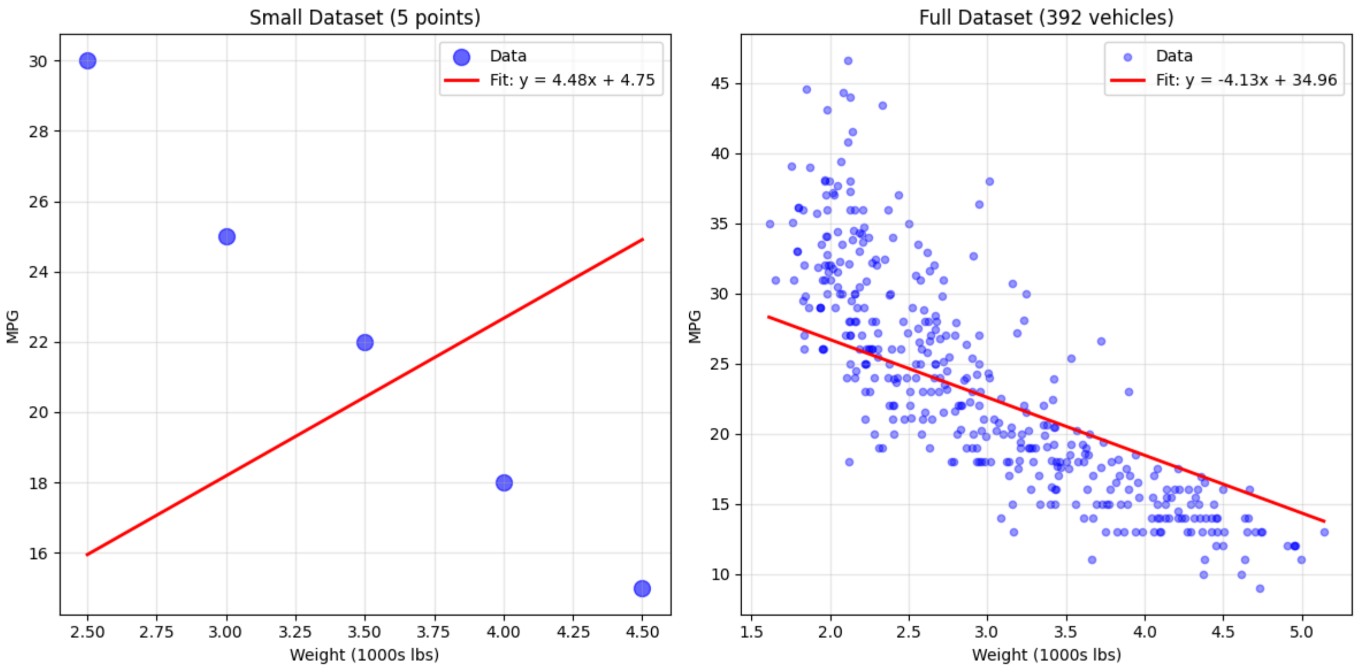

```
Standard GD final loss: 28.2222
Momentum GD final loss: 18.6766
```

In [8]:
```python
# Part 3.1: Learning curve
plt.figure(figsize=(10, 5))
plt.plot(model_full.loss_history)
plt.xlabel('Iteration')
plt.ylabel('Mean Squared Error (Loss)')
plt.title('Learning Curve: Loss vs Iteration')
plt.grid(True)
plt.show()
```



In [9]:
```python
# Part 3.2: Model fit visualization
plt.figure(figsize=(12, 6))

# Plot 1: Small dataset
plt.subplot(1, 2, 1)
plt.scatter(X_small, y_small, color='blue', s=100, alpha=0.6, label='Data')
# Plot the fitted line
x_line = np.linspace(min(X_small), max(X_small), 100)
```

```
y_line = [model.predict(x) for x in x_line]
plt.plot(x_line, y_line, 'r-', linewidth=2, label=f'Fit: y = {model.m:.2f}x + {model.
plt.xlabel('Weight (1000s lbs)')
plt.ylabel('MPG')
plt.title('Small Dataset (5 points)')
plt.legend()
plt.grid(True, alpha=0.3)

# Plot 2: Full dataset
plt.subplot(1, 2, 2)
plt.scatter(X_full, y_full, color='blue', s=20, alpha=0.4, label='Data')
x_line = np.linspace(min(X_full), max(X_full), 100)
y_line = [model_full.predict(x) for x in x_line]
plt.plot(x_line, y_line, 'r-', linewidth=2,
         label=f'Fit: y = {model_full.m:.2f}x + {model_full.b:.2f}')
plt.xlabel('Weight (1000s lbs)')
plt.ylabel('MPG')
plt.title(f'Full Dataset ({len(X_full)} vehicles)')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



Homework 3 Reflection Answers

- **What is the interpretation of the slope m?** For the full Auto MPG model, `m ≈ -4.1255`, so for every +1000 lbs increase in vehicle weight, predicted MPG *decreases* by about `4.13 MPG`.
- **Does the relationship make sense? Why or why not?** Yes. Heavier vehicles generally need more energy to accelerate and maintain speed, so they usually get lower fuel economy.
- **If a vehicle weighs 3000 lbs (x = 3.0), what MPG does your model predict?** The model predicts `22.5832 MPG`.
- **Did you need to adjust the learning rate from what worked on the small dataset? Why or why not?** No— `α = 0.01` was kept for both datasets. The full-dataset loss decreases smoothly (monotonic for standard GD), so it is stable, though convergence is slower than on the small dataset.

- **Which method converged faster (reached lower loss in fewer iterations)?** Gradient descent with momentum converged faster and reached much lower loss earlier.
- **Is the convergence of the momentum method smoother or more oscillatory?** It is faster but mildly more oscillatory near the minimum, while standard GD is smoother and steadily decreasing.
- **What final loss does each method achieve?** Standard GD: `28.2222`. Momentum GD: `18.6766`.
- **Does the loss decrease consistently?** For standard GD, yes (consistently decreasing). For momentum GD, it mostly decreases but has small oscillations.
- **At approximately what iteration does the loss appear to stabilize?** The standard GD curve flattens late (roughly around iterations `800–1000`), while momentum becomes nearly flat much earlier (roughly around `200–300`).
- **Why does the curve flatten out rather than reaching zero?** Real-world MPG data has noise and unmodeled factors (engine, aerodynamics, driving conditions, etc.), and a single straight line cannot perfectly fit all points. Therefore the minimum MSE is above zero.
- **How well does the line fit the data for the small dataset?** It fits reasonably well and captures the clear linear trend in the small synthetic points.
- **For the full dataset, describe the scatter around the line. Is it uniform or does it vary?** Scatter varies by weight range (heteroscedastic-like behavior): lighter vehicles show wider spread, while heavier vehicles are generally tighter around the line.
- **Looking at the full dataset plot, are there any outliers (points far from the line)?** Yes, there are several noticeable outliers (especially some light vehicles with much higher MPG than predicted).
- **Do you think a linear model is appropriate for this data? Why or why not?** A linear model is a good first baseline because it captures the main negative trend between weight and MPG, but it is not fully sufficient for best accuracy; nonlinear effects and additional features would improve fit.