Data Science I

Overall Points: 75 / 100

Exercise 6: Linear Algebra & Linear Regression

Submission Deadline: January 12 2026, 07:00 UTC
University of Oldenburg
Winter 2025/2026
Instructors: Jannik Schröder, Wolfram "Wolle" Wingerath

Submitted by: <Owiti, Horata >

Part 1: Linear Algebra

35 / 50

1.) Give a pair of square matrices A and B such that:

a) $AB = BA$ **(it commutes)**

Solution:

Let

A =

$$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

B =

$$\begin{bmatrix} 6 & 0 \\ 0 & 5 \end{bmatrix}$$

AB=

$$\begin{bmatrix} 12 & 0 \\ 0 & 15 \end{bmatrix}$$

✓

BA=

$$\begin{bmatrix} 12 & 0 \\ 0 & 15 \end{bmatrix}$$

AB=BA therefore commutates

b) $AB \neq BA$ **(it does not commute)**

Solution:

Let A =

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

B=

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

AB=

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

BA=

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

AB≠BA does not commutate

2.) Are the matrices MM^T **nd** M^TM **square and symmetric? Please explain.**

Solution:

Yes, the matrices are always square and symmetric.

To check if MM^{T} is a square, its matrix M = $(n \cdot m)$ and M^{T} $(n \cdot m)$ results into $(n \cdot n)$ which is a square and for $M^{\mathrm{T}}M$, its matrix M^{T} = $(m \cdot n)$ $(n \cdot m)$ results into $(m \cdot m)$ which is a square

To check for Symmetry For MM^{T} ; $(MM^{\mathrm{T}})^{\mathrm{T}} = (M^{\mathrm{T}})^{\mathrm{T}}M^{\mathrm{T}} = MM^{\mathrm{T}}$

For$M^{\mathrm{T}}M$; $(M^{\mathrm{T}}M)^{\mathrm{T}} = M^{\mathrm{T}}(M^{\mathrm{T}})^{\mathrm{T}} = M^{\mathrm{T}}M$

3.) Let A be a feature matrix, w be a weight vector, and b be the target vector, so that matrix equation $Aw = b$ holds. Please explain how to compute w with linear algebra. Does it make a difference whether A is a square and invertible matrix? Why or why not?

Solution:

Aw means that each row of A is multiplied with w to give one predicted value ..... ✗

4.) Please implement a function that computes the result of multiplying two matrices. Compare the speed of a library function for matrix multiplication to your own implementation.

a) How much faster is the library on products of random n × n matricies, as a function of n as n gets large?

Solution:

In [2]:
```
import numpy as np
import time

def matmul_naive(A, B):
    n, m = A.shape
    m2, p = B.shape
    assert m == m2
    C = np.zeros((n, p), dtype=float)
    for i in range(n):
        for j in range(p):
            s = 0.0
            for k in range(m):
                s += A[i, k] * B[k, j]
            C[i, j] = s
    return C

def benchmark_square(ns=(100, 200, 300, 400, 500)):
    results = []
    for n in ns:
        A = np.random.rand(n, n)
        B = np.random.rand(n, n)

        t0 = time.perf_counter()
        C1 = matmul_naive(A, B)
        t1 = time.perf_counter()

        t2 = time.perf_counter()
        C2 = A @ B
        t3 = time.perf_counter()
```

```
        assert np.allclose(C1, C2, atol=1e-8)

        naive_time = t1 - t0
        numpy_time = t3 - t2
        speedup = naive_time / numpy_time

        results.append((n, naive_time, numpy_time, speedup))
    return results

print(benchmark_square())
```

```
[(20, 0.004139457974815741, 0.0038855409948155284, 1.065349195990729),
(40, 0.02042841599904932, 4.041698412038386e-05, 505.44137430448393), (6
0, 0.047660417010774836, 2.9791990527883172e-05, 1599.77283042461), (80,
0.08621050001238473, 6.170800770632923e-05, 1397.0715182163035), (100,
0.16643537499476224, 0.0011534999939613044, 144.2872786008402)]
```

b) What about the product of an n × m and m × n matrix, where n << m?

Solution:

In [4]:
```
def benchmark_rectangular(n_values=(10, 20, 30), m_multiplier=100):
    results = []
    for n in n_values:
        m = n * m_multiplier  # m is much larger than n
        A = np.random.rand(n, m)  # n × m matrix
        B = np.random.rand(m, n)  # m × n matrix

        t0 = time.perf_counter()
        C1 = matmul_naive(A, B)
        t1 = time.perf_counter()

        t2 = time.perf_counter()
        C2 = A @ B
        t3 = time.perf_counter()

        assert np.allclose(C1, C2, atol=1e-8)

        naive_time = t1 - t0
        numpy_time = t3 - t2
        speedup = naive_time / numpy_time

        results.append((n, m, naive_time, numpy_time, speedup))
    return results

print(benchmark_rectangular())
```

```
[(10, 1000, 0.02686062501743436, 0.00013983299140818417, 192.09075588625
544), (20, 2000, 0.14394954199087806, 0.0011086660088039935, 129.8403133
5656076), (30, 3000, 0.45764037501066923, 0.00019087499822489917, 2397.5
92032831103)]
```

c) Now make sure that your implementation calculates C = A · B by first transposing B internally, so that all dot products are computed along rows of the matrices. Does that improve performance compared with an implementation that does not transpose B first? By how much?

Solution:

In [18…

```python
import numpy as np
import time

def matmul_naive(A, B):
    n, m = A.shape
    m2, p = B.shape
    assert m == m2
    C = np.zeros((n, p), dtype=float)
    for i in range(n):
        for j in range(p):
            s = 0.0
            for k in range(m):
                s += A[i, k] * B[k, j]
            C[i, j] = s
    return C

def matmul_with_transpose(A, B):
    n, m = A.shape
    m2, p = B.shape
    assert m == m2
    C = np.zeros((n, p), dtype=float)
    # Transpose B
    B_T = B.T
    for i in range(n):
        for j in range(p):
            s = 0.0
            for k in range(m):
                s += A[i, k] * B_T[j, k]  # indices change with transpos
            C[i, j] = s
    return C

def benchmark_transpose_comparison(ns=(100, 200, 300)):
    results = []
    for n in ns:
        A = np.random.rand(n, n)
        B = np.random.rand(n, n)

        # Time the naive implementation
        t0 = time.perf_counter()
        C1 = matmul_naive(A, B)
        t1 = time.perf_counter()
        naive_time = t1 - t0

        # Time the implementation with transpose
        t2 = time.perf_counter()
        C2 = matmul_with_transpose(A, B)
        t3 = time.perf_counter()
        transpose_time = t3 - t2
```

```
        # Time NumPy's implementation for reference
        t4 = time.perf_counter()
        C3 = A @ B
        t5 = time.perf_counter()
        numpy_time = t5 - t4

        # Verify all implementations give the same result
        assert np.allclose(C1, C2, atol=1e-8)
        assert np.allclose(C1, C3, atol=1e-8)



        results.append((n, naive_time, transpose_time, numpy_time,))
    return results

print(benchmark_transpose_comparison())
```

missing answear sentence -5

```
[(100, 0.19429195899283513, 0.16745750000700355, 0.0001834580034483224
2), (200, 1.3465008329949342, 1.3584501250006724, 0.0005222125017596408
7), (300, 4.636155791988131, 4.5665406250045635, 0.002562917012255639)]
```

Part 2: Linear Regression

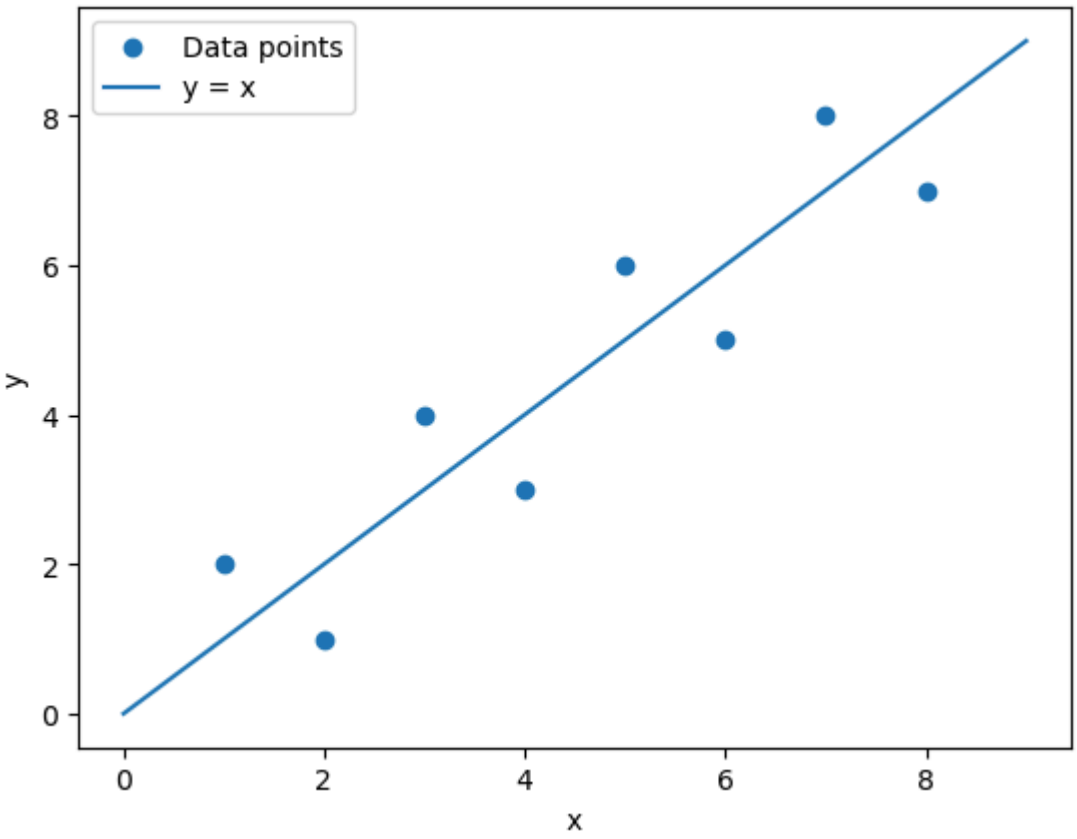
40 / 50

5.) Construct an example on n ≥ 6 points where the optimal regression line is y = x, even though none of the input points lie directly on this line. Please visualize your example with a plot.

Solution:

In [20…

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

points = np.array([[1,2],[2,1],[3,4],[4,3],[5,6],[6,5],[7,8],[8,7]])
x = points[:,0]
y = points[:,1]

plt.scatter(x, y, label="Data points")
plt.plot([0,9], [0,9], label="y = x")  # the regression line target
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```



6.) Suppose we fit a regression line to predict the shelf life of an apple based on its weight. For a particular apple, we predict the shelf life to be 4.6 days. The apples residual is −0.6 days. Did we over or under estimate the shelf-life of the apple? Please explain your reasoning.

Solution:

In [19…

```
# Given:
predicted = 4.6      # y_hat
residual = -0.6      # r = y - y_hat

# residual = actual - predicted  =>  actual = predicted + residual
actual = predicted + residual

print(f"Predicted shelf life (y_hat): {predicted} days")
print(f"Residual (y - y_hat): {residual} days")
print(f"Actual shelf life (y): {actual} days")

if predicted > actual:
    print("Conclusion: We OVERestimated the shelf life.")
elif predicted < actual:
    print("Conclusion: We UNDERESTIMATED the shelf life.")
else:
    print("Conclusion: Perfect prediction (no error).")
```

```
Predicted shelf life (y_hat): 4.6 days
Residual (y - y_hat): -0.6 days
Actual shelf life (y): 3.9999999999999996 days
Conclusion: We OVERestimated the shelf life.
```

✓

7.) Suppose we want to find the best-fitting function $f(x) = y = w^2x + wx$. How can we use linear regression to find the best value of w? Please explain.

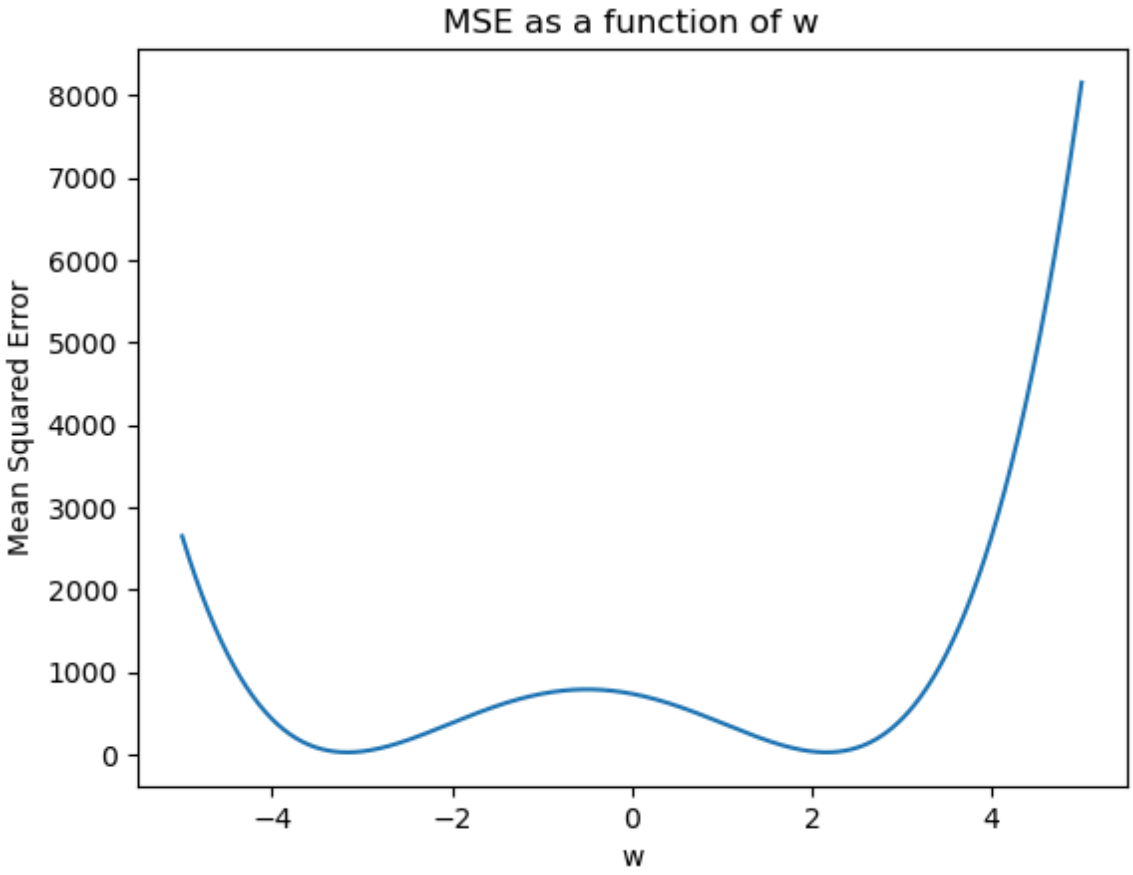
Solution:

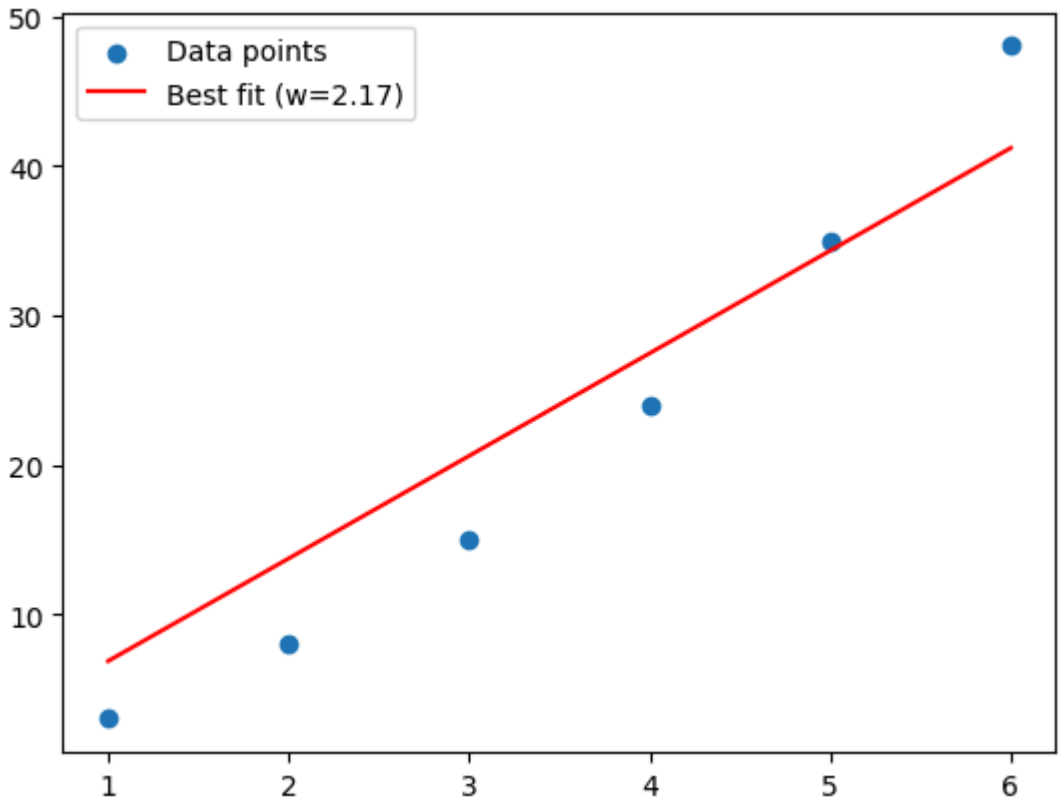
In [21…

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

# Example data (x, y)
x = np.array([1, 2, 3, 4, 5, 6])
y = np.array([3, 8, 15, 24, 35, 48])

# Model: y = w^2 x + w x
def predict(x, w):
    return w**2 * x + w * x

# Mean Squared Error
def mse(y, y_pred):
    return np.mean((y - y_pred)**2)

# Try many values of w
w_values = np.linspace(-5, 5, 1000)
errors = np.array([mse(y, predict(x, w)) for w in w_values])

# Best w
best_w = w_values[np.argmin(errors)]

# Output
print("Best w:", best_w)
print("Minimum MSE:", np.min(errors))

# Plot error vs w
plt.plot(w_values, errors)
plt.xlabel("w")
plt.ylabel("Mean Squared Error")
plt.title("MSE as a function of w")
plt.show()

# Plot best fit
plt.scatter(x, y, label="Data points")
plt.plot(x, predict(x, best_w), color="red", label=f"Best fit (w={best_w
plt.legend()
plt.show()
```

```
Best w: 2.1671671671671673
Minimum MSE: 22.979071347060636
```

✗



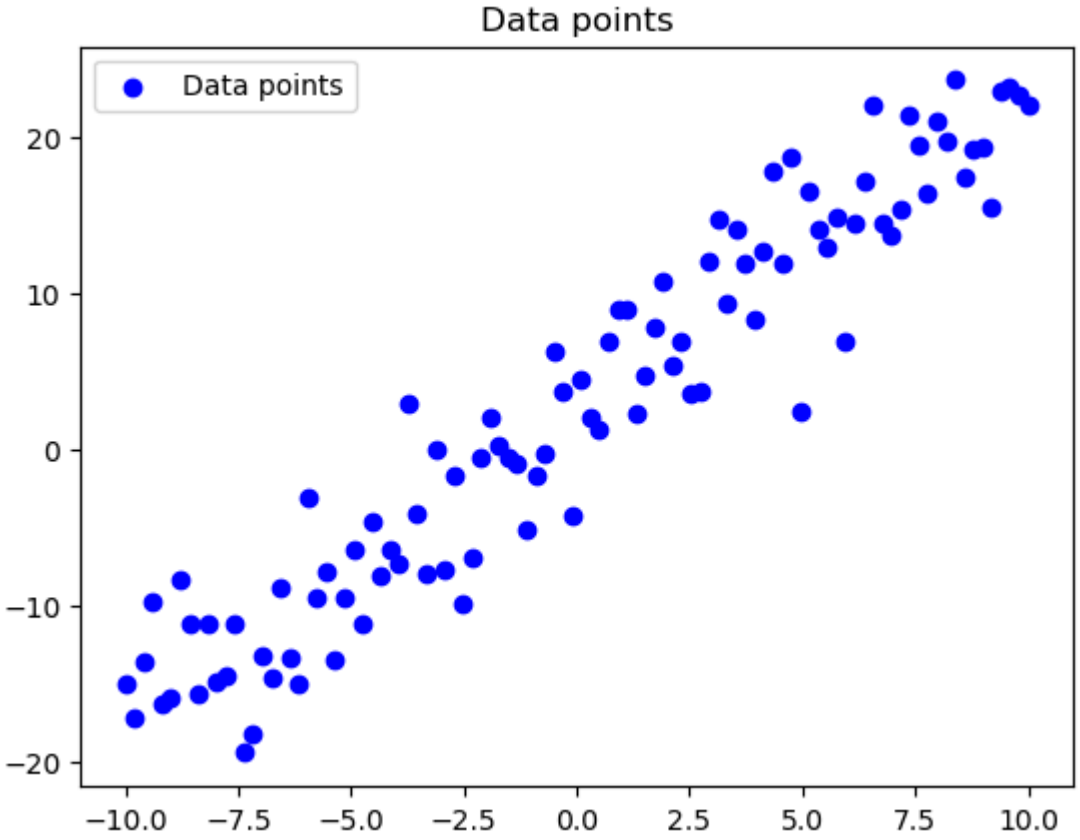


8.) Consider the following data set:

In [32…

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

```
np.random.seed(42)

# Parameters of the line
m = 2
b = 3

# Number of data points
n = 100

# Generate data points
x = np.linspace(-10, 10, n)
y = m * x + b + np.random.normal(0, 4, n)

# Plot the data points
plt.scatter(x, y, color='blue', label='Data points')
plt.title('Data points')
plt.legend()
plt.show()
```



Assume that the data points have been generated by a linear function $y = mx + b$ with some random noise added to the y values. Please implement the gradient descent algorithm to estimate the parameters m (slope) and b (intercept) of the linear function via the following steps.

1. **Plot the provided dataset on a scatter plot.**
2. **Implement the gradient descent algorithm. Use the mean squared error as the cost function.**
3. **For each iteration of the gradient descent algorithm, plot the line with the current estimate of m and b on the scatter plot with the data points. You should end up with a plot where lines of different colors represent the**

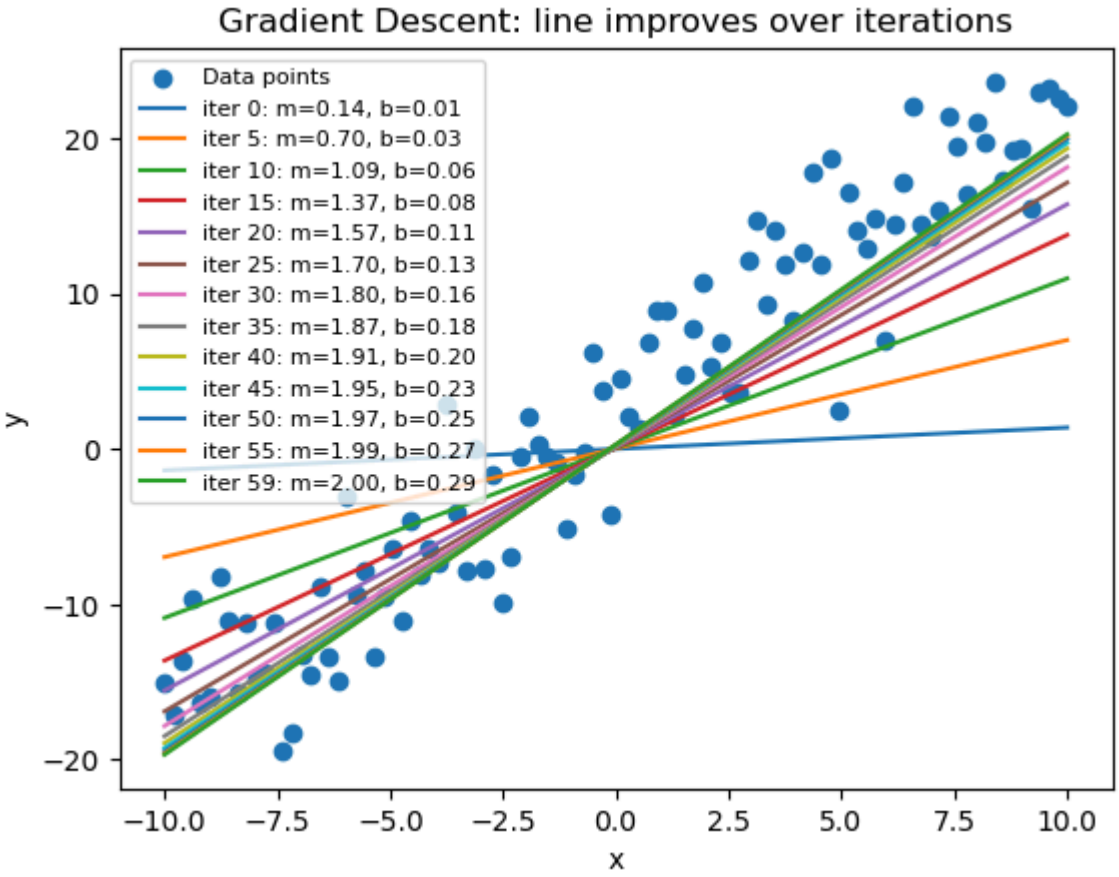
estimated line of best fit at different stages of the algorithm. This will help to visualize how the line of best fit improves with each iteration.

Solution:

In [22…

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

# ----------------------------
# 1) Generate the dataset (given in the sheet)
# ----------------------------
np.random.seed(42)

m_true = 2
b_true = 3
n = 100

x = np.linspace(-10, 10, n)
y = m_true * x + b_true + np.random.normal(0, 4, n)

# ----------------------------
# 2) Gradient Descent settings
# ----------------------------
m = 0.0
b = 0.0
alpha = 0.001       # learning rate
iterations = 60     # number of steps

# Mean Squared Error
def mse(y, y_pred):
    return np.mean((y - y_pred) ** 2)

# ----------------------------
# 3) Run Gradient Descent + plot lines over time
# ----------------------------
plt.figure()
plt.scatter(x, y, label="Data points")

for t in range(iterations):
    y_pred = m * x + b

    # Gradients for MSE J(m,b) = (1/n) * sum (y - (mx + b))^2
    dm = (-2 / n) * np.sum(x * (y - y_pred))
    db = (-2 / n) * np.sum(y - y_pred)

    # Update parameters
    m = m - alpha * dm
    b = b - alpha * db

    # Plot the current line every few iterations (to avoid too many line
    if t % 5 == 0 or t == iterations - 1:
        plt.plot(x, m * x + b, label=f"iter {t}: m={m:.2f}, b={b:.2f}")

plt.title("Gradient Descent: line improves over iterations")
plt.xlabel("x")
plt.ylabel("y")
```

```
plt.legend(fontsize=8)
plt.show()

print("Final estimated m:", m)
print("Final estimated b:", b)
print("Final MSE:", mse(y, m * x + b))
```




```
Final estimated m: 1.997968407985234
Final estimated b: 0.29254244691517245
Final MSE: 18.321898945347083
```

Finally: Submission

Save your notebook and submit it (as both **notebook and PDF file**). And please don't forget to ...

- ... choose a **file name** according to convention (see Exercise Sheet 1, but please **add your group name as a suffix** like `_group01`) and to
- ... include the **execution output** in your submission!