Lab 3 solution key

1.

```
import numpy as np
import scipy.linalg as la

# Define a square matrix A
A = np.array([[2, 1, 5], [4, 4, -4], [1, 3, 1]], dtype=float)

# Perform LU decomposition (P * A = L * U)
P, L, U = la.lu(A)

print("Permutation Matrix P:\n", P)
print("Lower Triangular Matrix L:\n", L)
print("Upper Triangular Matrix U:\n", U)

# Verification: P * A = L * U
PA = P @ A
LU = L @ U

print("\nVerification (P * A == L * U):", np.allclose(PA, LU))
```

2.

```
import numpy as np
import scipy.linalg as la

# --- Method 1: SciPy LU + Diagonal Extraction ---
A = np.array([[2, 4, 2], [1, 5, 3], [4, 1, 6]], dtype=float)

P, L, U_scipy = la.lu(A)
D_diag = np.diag(U_scipy)
D = np.diag(D_diag)
U = U_scipy / D_diag[:, None]

print("--- Method 1: Using SciPy ---")
print("L:\n", L)
print("D:\n", D)
print("U:\n", U)

# --- Method 2: Manual Step-by-Step Gaussian Elimination ---
n = A.shape[0]
L_manual = np.eye(n)
U_manual = A.copy().astype(float)
```

```
for i in range(n):
    for j in range(i + 1, n):
        factor = U_manual[j, i] / U_manual[i, i]
        L_manual[j, i] = factor
        U_manual[j, :] -= factor * U_manual[i, :]

D_manual = np.diag(np.diag(U_manual))
U_manual = U_manual / np.diag(U_manual)[:, None]

print("\n--- Method 2: Manual Gaussian Elimination ---")
print("L:\n", L_manual)
print("D:\n", D_manual)
print("U:\n", U_manual)
```

3.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

# Input data points
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype=float)
Y = np.array([2, 3, 5, 7, 11, 13, 17, 19, 23, 29], dtype=float)

# --- 1. From Scratch using Gradient Descent ---
m = 0.0  # slope
c = 0.0  # intercept
alpha = 0.01  # learning rate
epochs = 2000
n = len(X)

for _ in range(epochs):
    Y_pred = m * X + c
    dm = (-2 / n) * np.sum(X * (Y - Y_pred))
    dc = (-2 / n) * np.sum(Y - Y_pred)
    m -= alpha * dm
    c -= alpha * dc

print(f"Gradient Descent -> Slope: {m:.4f}, Intercept: {c:.4f}")

# --- 2. Using scikit-learn ---
X_2d = X.reshape(-1, 1)
model = LinearRegression()
model.fit(X_2d, Y)
```

```
print(
    f"Scikit-Learn     -> Slope: {model.coef_[0]:.4f}, Intercept: {model.intercept_:.4f}"
)

# Plotting
plt.scatter(X, Y, color="red", label="Data Points")
plt.plot(X, m * X + c, color="blue", label="GD Fit")
plt.plot(
    X,
    model.predict(X_2d),
    color="green",
    linestyle="--",
    label="Sklearn Fit",
)
plt.title("Linear Regression Comparison")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```

4.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression

# Synthetic dataset: X1, X2, Y
X1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
X2 = np.array([2, 1, 4, 2, 5, 3, 6, 7])
Y = np.array([5, 6, 10, 9, 14, 12, 18, 20])

# (a) From scratch: Design matrix & Normal Equation
X_design = np.column_stack([np.ones(len(X1)), X1, X2])
beta = np.linalg.inv(X_design.T @ X_design) @ X_design.T @ Y

intercept, coeff_X1, coeff_X2 = beta[0], beta[1], beta[2]
Y_hat = X_design @ beta

print(
    f"Scratch -> Intercept: {intercept:.4f}, Beta_X1: {coeff_X1:.4f}, Beta_X2: {coeff_X2:.4f}"
)

# (b) R-squared calculation
ss_res = np.sum((Y - Y_hat) ** 2)
ss_tot = np.sum((Y - np.mean(Y)) ** 2)
```

```
r2_scratch = 1 - (ss_res / ss_tot)
print(f"Scratch R² Score: {r2_scratch:.4f}")

# (c) Verification using Scikit-Learn
X_features = np.column_stack([X1, X2])
sk_model = LinearRegression()
sk_model.fit(X_features, Y)

print(
    f"Sklearn -> Intercept: {sk_model.intercept_:.4f}, Coeffs: {sk_model.coef_}"
)
print(f"Sklearn R² Score: {sk_model.score(X_features, Y):.4f}")

# (d) 3D Surface Plot
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(X1, X2, Y, color="red", label="Data Points")

# Generate Grid for Plane
x1_surf, x2_surf = np.meshgrid(
    np.linspace(min(X1), max(X1), 10), np.linspace(min(X2), max(X2), 10)
)
y_surf = intercept + coeff_X1 * x1_surf + coeff_X2 * x2_surf

ax.plot_surface(x1_surf, x2_surf, y_surf, alpha=0.5, cmap="coolwarm")
ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("Y")
plt.title("Multiple Linear Regression Fit Plane")
plt.show()
```

5.

```
import numpy as np
# Define non-square m x n matrix A
A = np.array([[1, 2, 3], [4, 5, 6]], dtype=float)

# Compute Moore-Penrose Pseudoinverse
A_plus = np.linalg.pinv(A)

print("Original Matrix A:\n", A)
print("\nPseudoinverse A+:\n", A_plus)

# Verify condition: A * A+ * A = A
verification = np.matmul(np.matmul(A, A_plus), A)
```

```
print("\nVerification (A * A+ * A == A):")
print(np.isclose(verification, A).all())
```