Lab Session: Introduction to Numerical Linear Algebra

Matrices, Vectors, Numpy, and Gaussian Elimination

Mohamed Amine Hamadi

February 9, 2026

University Mohammed VI Polytechnic

THE UM6P VANGUARD CENTER

#Empowering Minds.

1. Getting Started: Setup

First, we need to import the necessary libraries.

- `numpy`: The core library for matrix operations.
- `matplotlib.pyplot`: For visualizing data (and matrices).

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

```

Note on Indexing:

- **Python starts counting at 0.**
- Matrix element at row 2, col 3 is `A[1, 2]`.

2. Creating Vectors and Matrices

Let's create some basic structures.

```
# 1. Define a specific matrix manually
A = np.array([[3, 2, -1],
[2, -2, 4],
[-2, -1, 2]])

# 2. Zeros matrix (useful for pre-allocating space)
Z = np.zeros((3, 3))

# 3. Identity matrix
I = np.eye(3)

# 4. Ones vector
b = np.ones((3, 1))

print("Matrix A:\n", A)
print("Vector b:\n", b)
```

3. Visualizing Matrices (The "Spy" Function)

Creating a Banded Matrix Directly Let's create a matrix with a specific diagonal pattern (0 on main, 1 above, -1 below) using loops.

The Challenge:

- Initialize a 50×50 matrix full of zeros.
- Use a `for` loop and `if` statements to set specific values:
  - Upper Diagonal ($j = i + 1$): Set value to **1**.
  - Lower Diagonal ($j = i - 1$): Set value to **-1**.
  - Main Diagonal: Leave it as **0**.

```
N = 50
A = np.zeros((N, N))  # Start with all zeros


```

Remark: After writing the code, run `plt.spy(A, markersize=5, c='blue')` to visualize your bands!

4. Implementing Gaussian Elimination

Let's code the algorithm we discussed in class.

1. **Forward Elimination:**
   - For each column k, calculate the multiplier for row i: $m = A_{ik}/A_{kk}$.
   - Update row i: $Row_i \leftarrow Row_i - m \times Row_k$.
2. **Backward Substitution:**
   - Start from the bottom (x_n) and move upwards.
   - For row i: $x_i = (b_i - \sum A_{ij}x_j)/A_{ii}$.

```
def gaussian_elimination(A, b):
n = len(b)
# --- 1. Forward Elimination ---
for k in range(n-1):
for i in range(k+1, n):
# --- 2. Backward Substitution ---
x = np.zeros((n, 1))

# Find the last variable x[n-1]
x[n-1] = ...
# Loop backwards from n-2 down to 0
return x
```

5. Testing your Function

Use a 3×3 example to verify your code.

```
# Define a system
A = np.array([[2., 1., -1.],
[-3., -1., 2.],
[-2., 1., 2.]])
b = np.array([[8.], [-11.], [-3.]])

# Call your function
x_ge = gaussian_elimination(A, b)

print("Solution from GaussianElimination Code:\n", x_ge)

```

Expected Result: $x_1 = 2, x_2 = 3, x_3 = -1.$

6. Built-in Functions

Python provides optimized libraries (NumPy and SciPy) for solving $Ax = b$

```
from scipy.linalg import lu
import numpy.linalg as LA

# 1. Get L and U matrices
# (We ignore the Pivot matrix P using '_' for this simple example)
_, L, U = lu(A)

# 2. Solve Ax = b using the L and U decomposition
# Step A: Forward substitution (Solve L*y = b)
y = LA.solve(L, b)

# Step B: Backward substitution (Solve U*x = y)
x_lu = LA.solve(U, y)

print("Solution using LU steps:\n", x_lu)

# 3. Solve using the Inverse operator (x = A^-1 * b)
# Note: '@' is the operator for matrix multiplication
A_inv = LA.inv(A)
```

7. Comparing Results

Let's look at the numbers directly to verify our work.

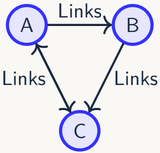
```
# 1. Print the solutions side-by-side to compare
print("Solution from Custom Code:\n", x_ge)
print("Solution from LU:\n", x_lu)
print("Solution from Inverse:\n", x_inv)

# 2. Check the Residual: A*x - b
# If we solved the system correctly, the result should be (approximately) a
 zero vector.


residual = A @ x_lu - b # try it also for x_ge and x_inv
print("\nResidual (A*x - b):\n", residual)

```

What to look for

- The 3 solution vectors should look identical.
- The Residual should look like `[ [0], [0], [0] ]`.
- You might see tiny numbers like 1.2×10^{-12} . These are computer rounding errors,

8. Exercise: Ranking the Mini-Internet

The Problem: We want to calculate the "Importance Score" for nodes A, B, and C using the graph from the lecture :



Your Task:

1. Based on the graph logic (Split votes, Full votes), create the Matrix M (where $x = M \cdot x$).
2. Create matrix $A = (I - M)$ and vector $b = [0, 0, 0]^T$.
3. **Trick:** Since we have infinite solutions, let's fix the **Total Score** to 1. Replace the **last row of** A with $[1, 1, 1]$ and the **last element of** b with 1.
4. Solve the system using your `gaussian_elimination` function.

9. Solution Hint

```
# We build M such that x = M * x
# Row i corresponds to equation for variable x_i

# Row 1 (for x_A): [0, 0, 1]  -> Matches eq 1 (1 * x_C)
# Row 2 (for x_B): [1, 0, 0]  -> Matches eq 2 (1 * x_A)
# Row 3 (for x_C): [0.5, 1, 0] -> Matches eq 3 (0.5*x_A + 1*x_B)

M = np.array([[0.0, 0.0, 1.0],
[1.0, 0.0, 0.0],
[0.5, 1.0, 0.0]])

# 2. Define A = I - M
I = np.eye(n)
A = I - M
b = np.zeros((n, 1))

# 3. Apply the Normalization Trick (Total Score = 1)
# Replace last row with coefficients of sum equation (x1+x2+x3 = 1)
A[n-1, :] = [1.0, 1.0, 1.0]
b[n-1, 0] = 1.0
# 4. Solving Part
```

10. Understanding the Result

The Scores we found:

- Node A: 0.40
- Node B: 0.20
- Node C: 0.40

What does this tell us?

1. **The "Link Loop":** Nodes A and C have the same score because they are linked in a circle (A -> ... -> C -> A). They support each other equally.
2. **Dilution of Power:** Node B has a lower score because its parent (Node A) is too busy! Node A links to both B and C, so it splits its vote in half.
3. **Stability:** Even though the logic is circular, Linear Algebra (Gaussian Elimination) found a stable, unique solution.

Congratulations! You just performed your first Google Search simulation!