Project 3 - 3x3 Matrix

Gaussian Elimination Report

John Akujobi
MATH 374: Scientific Computation (Spring 2025), South Dakota State University
Professor: Dr Kimn, Dept. Of Math & Statistics
GitHub: [jakujobi]

Problem Statement

Matrix A:

```
[[3. 1. 2.]
 [1. 2. 0.]
 [0. 1. 1.]]
```

Vector b:

```
[10.  8.  3.]
```

Algorithm Overview

- Compute scale factors s[i] = max_j |A[i, j]|
- For each column k:
    1. Compute ratio |A[i, k]|/s[i] for i=k.. n-1
    2. Select pivot row with max ratio, swap if needed
    3. Eliminate A[i, k] for i>k
- Back-substitution to solve for x

```python
def scaled_partial_pivot_gauss(A, b, return_steps=False, tol=1e-10):
    """
    Solves Ax = b using Gaussian elimination with scaled partial pivoting.
    Returns the solution vector x, and optionally step-by-step logs.

    Parameters:
    -----------
    A : array-like
        Coefficient matrix
    b : array-like
        Right-hand side vector
    return_steps : bool, optional
        If True, return detailed steps of the algorithm
    tol : float, optional
        Tolerance for detecting near-singular matrices

    Returns:
    --------
    x : ndarray
```

```python
            Solution vector
        steps : list, optional
            Detailed steps of the algorithm (if return_steps=True)
        """
        # Convert inputs to numpy arrays
        A = np.array(A, dtype=float)
        b = np.array(b, dtype=float)
        n = A.shape[0]
        # Validate dimensions
        if A.shape[0] != A.shape[1]:
            raise ValueError("Matrix A must be square.")
        if b.size != n:
            raise ValueError("Vector b length must equal A dimension.")

        steps = []
        # Compute scaling factors for each row
        s = np.max(np.abs(A), axis=1)

        # Check for zero scaling factors
        if np.any(s == 0):
            raise ValueError("Matrix contains a row of zeros.")

        # Forward elimination with scaled partial pivoting
        for k in range(n - 1):
            # Determine pivot row based on scaled ratios
            ratios = np.abs(A[k:, k]) / s[k:]
            idx_max = np.argmax(ratios)
            p = k + idx_max
            ratio = float(ratios[idx_max])  # scaled ratio for pivot

            # Check for near-singular matrix
            if abs(A[p, k]) < tol:
                raise ValueError("Matrix is singular or nearly singular.")

            # Swap rows if necessary, logging ratio
            if p != k:
                A[[k, p], :] = A[[p, k], :]
                b[k], b[p] = b[p], b[k]
                steps.append({
                    "step": "swap",
                    "k": k,
                    "pivot_row": p,
                    "ratio": ratio,
                    "A": A.copy(),
                    "b": b.copy()
                })
            else:
                steps.append({
                    "step": "pivot",
                    "k": k,
                    "pivot_row": p,
                    "ratio": ratio,
                    "A": A.copy(),
                    "b": b.copy()
                })
            # Eliminate entries below pivot
            for i in range(k + 1, n):
                # Compute multiplier with fraction components
                num = A[i, k]
                den = A[k, k]
                m = num / den
```

```
            # Perform elimination row update
            A[i, k:] = A[i, k:] - m * A[k, k:]
            b[i] = b[i] - m * b[k]
            steps.append({
                "step": "elimination",
                "k": k,
                "i": i,
                "multiplier": m,
                "mult_num": num,
                "mult_den": den,
                "A": A.copy(),
                "b": b.copy()
            })

    # Back substitution to solve for x
    x = np.zeros(n, dtype=float)
    for i in reversed(range(n)):
        if abs(A[i, i]) < tol:
            raise ValueError("Matrix is singular or nearly singular.")
        x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]
        steps.append({
            "step": "back_substitution",
            "i": i,
            "value": x[i],
            "A": A.copy(),
            "b": b.copy()
        })

    if return_steps:
        return x, steps
    return x
```

Step-by-step Details

Step 1: Pivot

Column 0: pivot row 0 selected with scaled ratio 1.000. No swap needed.

```
[3.0, 1.0, 2.0, 10.0]
[1.0, 2.0, 0.0, 8.0]
[0.0, 1.0, 1.0, 3.0]
```

Step 2: Elimination

Row 1: eliminate A[1,0] using multiplier 1.0/3.0 = 0.333.

```
[3.0, 1.0, 2.0, 10.0]
[0.0, 1.6666666666666667, -0.6666666666666666, 4.666666666666667]
[0.0, 1.0, 1.0, 3.0]
```

Step 3: Elimination

Row 2: eliminate A[2,0] using multiplier 0.0/3.0 = 0.000.

```
[3.0, 1.0, 2.0, 10.0]
[0.0, 1.6666666666666667, -0.6666666666666666, 4.666666666666667]
```

```
[0.0, 1.0, 1.0, 3.0]
```

Step 4: Swap

Column 1: pivot row 2 selected with scaled ratio 1.000. Swapped row 1 and 2.

```
[3.0, 1.0, 2.0, 10.0]
[0.0, 1.0, 1.0, 3.0]
[0.0, 1.6666666666666667, -0.6666666666666666, 4.666666666666667]
```

Step 5: Elimination

Row 2: eliminate A[2,1] using multiplier 1.6666666666666667/1.0 = 1.667.

```
[3.0, 1.0, 2.0, 10.0]
[0.0, 1.0, 1.0, 3.0]
[0.0, 0.0, -2.3333333333333335, -0.33333333333333304]
```

Step 6: Back Substitution

Back substitute for x[2]: x[2] = 0.142857.

```
[3.0, 1.0, 2.0, 10.0]
[0.0, 1.0, 1.0, 3.0]
[0.0, 0.0, -2.3333333333333335, -0.33333333333333304]
```

Step 7: Back Substitution

Back substitute for x[1]: x[1] = 2.85714.

```
[3.0, 1.0, 2.0, 10.0]
[0.0, 1.0, 1.0, 3.0]
[0.0, 0.0, -2.3333333333333335, -0.33333333333333304]
```

Step 8: Back Substitution

Back substitute for x[0]: x[0] = 2.28571.

```
[3.0, 1.0, 2.0, 10.0]
[0.0, 1.0, 1.0, 3.0]
[0.0, 0.0, -2.3333333333333335, -0.33333333333333304]
```

Solution

```
(2.285714285714286, 2.857142857142857, 0.1428571428571427)
```

Performance Metrics

Execution Time: 0.000239 seconds
Estimated Floating-point Operations: 18

Solution Verification

Residual (Ax - b): [0. 0. 0.]
Infinity Norm of Residual: 0.000 e+00

References & Notes

- [Gaussian elimination – Wikipedia]
- Burden & Faires, *Numerical Analysis*, Ch. 3
- Cheney & Kincaid, *Numerical Mathematics and Computing*, 7 th Edition
- Uses scaled partial pivoting for numerical stability.
- Debugging assistance from Qwen 3 locally run