Project 3 - 4x4 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. -13.   9.   3.]
 [ -6.   4.   1. -18.]
 [  6.  -2.   2.   4.]
 [ 12.  -8.   6.  10.]]
```

Vector b:

```
[-19. -34.  16.  26.]
```

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:
    --------
```

```python
        x : ndarray
            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: Swap

Column 0: pivot row 2 selected with scaled ratio 1.000. Swapped row 0 and 2.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[-6.0, 4.0, 1.0, -18.0, -34.0]
[3.0, -13.0, 9.0, 3.0, -19.0]
[12.0, -8.0, 6.0, 10.0, 26.0]
```

Step 2: Elimination

Row 1: eliminate A[1,0] using multiplier -6.0/6.0 = -1.000.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, 2.0, 3.0, -14.0, -18.0]
[3.0, -13.0, 9.0, 3.0, -19.0]
[12.0, -8.0, 6.0, 10.0, 26.0]
```

Step 3: Elimination

Row 2: eliminate A[2,0] using multiplier 3.0/6.0 = 0.500.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, 2.0, 3.0, -14.0, -18.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[12.0, -8.0, 6.0, 10.0, 26.0]
```

Step 4: Elimination

Row 3: eliminate A[3,0] using multiplier 12.0/6.0 = 2.000.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, 2.0, 3.0, -14.0, -18.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, -4.0, 2.0, 2.0, -6.0]
```

Step 5: Swap

Column 1: pivot row 2 selected with scaled ratio 2.000. Swapped row 1 and 2.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 2.0, 3.0, -14.0, -18.0]
[0.0, -4.0, 2.0, 2.0, -6.0]
```

Step 6: Elimination

Row 2: eliminate A[2,1] using multiplier 2.0/-12.0 = -0.167.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 0.0, 4.333333333333333, -13.833333333333334, -22.5]
[0.0, -4.0, 2.0, 2.0, -6.0]
```

Step 7: Elimination

Row 3: eliminate A[3,1] using multiplier -4.0/-12.0 = 0.333.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 0.0, 4.333333333333333, -13.833333333333334, -22.5]
[0.0, 0.0, -0.6666666666666665, 1.6666666666666667, 3.0]
```

Step 8: Pivot

Column 2: pivot row 2 selected with scaled ratio 0.722. No swap needed.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 0.0, 4.333333333333333, -13.833333333333334, -22.5]
[0.0, 0.0, -0.6666666666666665, 1.6666666666666667, 3.0]
```

Step 9: Elimination

Row 3: eliminate A[3,2] using multiplier -0.6666666666666665/4.333333333333333 = -0.154.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
```

```
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 0.0, 4.333333333333333, -13.833333333333334, -22.5]
[0.0, 0.0, 0.0, -0.46153846153846145, -0.46153846153846123]
```

Step 10: Back Substitution

Back substitute for x[3]: x[3] = 1.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 0.0, 4.333333333333333, -13.833333333333334, -22.5]
[0.0, 0.0, 0.0, -0.46153846153846145, -0.46153846153846123]
```

Step 11: Back Substitution

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

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 0.0, 4.333333333333333, -13.833333333333334, -22.5]
[0.0, 0.0, 0.0, -0.46153846153846145, -0.46153846153846123]
```

Step 12: Back Substitution

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

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 0.0, 4.333333333333333, -13.833333333333334, -22.5]
[0.0, 0.0, 0.0, -0.46153846153846145, -0.46153846153846123]
```

Step 13: Back Substitution

Back substitute for x[0]: x[0] = 3.

```
[6.0, -2.0, 2.0, 4.0, 16.0]
[0.0, -12.0, 8.0, 1.0, -27.0]
[0.0, 0.0, 4.333333333333333, -13.833333333333334, -22.5]
[0.0, 0.0, 0.0, -0.46153846153846145, -0.46153846153846123]
```

Solution

```
(3.0000000000000004, 0.9999999999999991, -2.0000000000000013, 0.9999999999999996)
```

Performance Metrics

Execution Time: 0.000780 seconds
Estimated Floating-point Operations: 42

Solution Verification

Residual (Ax - b): [0.00000000 e+00 0.00000000 e+00 3.55271368 e-15 3.55271368 e-15]
Infinity Norm of Residual: 3.553 e-15

References & Notes

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