## **Gaussian Elimination with Scaled Partial Pivoting**

Use the sidebar to switch between examples and the Interactive Playground.

## **4×4 Example Report**

#### John Akujobi

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

### **Problem Statement**

Solve the linear system Ax = b, where:

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

```
[-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 find solution vector x

```
for k in range(n-1):
    s = [max(abs(A[i,:])) for i in range(n)]
    ratios = [abs(A[i,k])/s[i] for i in range(k,n)]
    pivot = k + int(np.argmax(ratios))
    if pivot != k:
        A[[k,pivot]] = A[[pivot,k]]
        b[k], b[pivot] = b[pivot], b[k]
    for i in range(k+1,n):
        m = A[i,k]/A[k,k]
        A[i,k:] -= m * A[k,k:]
# Back-substitution...
```

## **Step-by-Step Summary**

	Step#	Туре	K	ī	plvet_rew	mu <sup>lti</sup> p <sup>li</sup> er	ra <sup>ti</sup> o	value
0	1	swap	0		2		1.0	

localhost:8501 1/14

#### Gaussian Elimination Demo

	Step#	Туре	k	į	pivot_row	multiplier	ratio	value
1	2	elimination	0	1		-1.0		
2	3	elimination	0	2		0.5		
3	4	elimination	0	3		2.0		
4	5	swap	1		2		2.0	
5	6	elimination	1	2		-0.1666666666666666		
6	7	elimination	1	3		0.333333333333333		
7	8	pivot	2		2		0.72222222222222	
8	9	elimination	2	3		-0.15384615384615383		
9	10	back_substitution		3				0.99999999999996
10	11	back_substitution		2				-2.0000000000000013
11	12	back_substitution		1				0.99999999999999
12	13	back_substitution		0				3.0000000000000004

### **Intermediate Matrices**

## Original Matrix (k=0)

	х0	x1	x2	х3
0	3	-13	9	3
1	-6	4	1	-18
2	6	-2	2	4
3	12	-8	6	10

## Original Matrix (k=1) After Pivot (k=1)

	x0	x1	x2	х3
0	3	-13	9	3
1	-6	4	1	-18
2	6	-2	2	4
3	12	-8	6	10

## Original Matrix (k=2)

	x0	x1	x2	хЗ
0	3	-13	9	3
1	-6	4	1	-18
2	6	-2	2	4
3	12	-8	6	10

## After Pivot (k=0)

	x0	x1	x2	х3
0	6	-2	2	4
1	-6	4	1	-18
2	3	-13	9	3
3	12	-8	6	10

	x0	x1	x2	х3
0	6	-2	2	4
1	0	-12	8	1
2	0	2	3	-14
3	0	-4	2	2

## After Pivot (k=2)

	x0	x1	x2	хЗ
0	6	-2	2	4
1	0	-12	8	1
2	0	0	4.3333	-13.8333
3	0	0	-0.6667	1.6667

## After Elimination (k=0)

	x0	x1	x2	х3
0	6	-2	2	4
1	0	2	3	-14
2	0	-12	8	1
3	0	-4	2	2

## After Elimination (k=1)

	x0	x1	x2	х3
0	6	-2	2	4
1	0	-12	8	1
2	0	0	4.3333	-13.8333
3	0	0	-0.6667	1.6667

## After Elimination (k=2)

	x0	x1	x2	х3
0	6	-2	2	4
1	0	-12	8	1
2	0	0	4.3333	-13.8333
3	0	0	0	-0.4615

### **Core Solver Code**

```
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
       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:</pre>
            raise ValueError("Matrix is singular or nearly singular.")
        # Swap rows if necessary, logging ratio
        if p != k:
            A[[k, p], :] = A[[p, k], :]
```

localhost:8501 3/14

```
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:</pre>
        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
```

#### **Performance Metrics**

localhost:8501 4/14

13

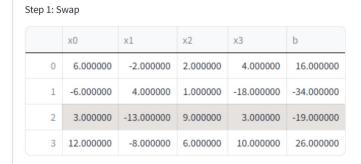
Execution Time: 0.000756 seconds

#### Estimated Floating-point Operations: 42

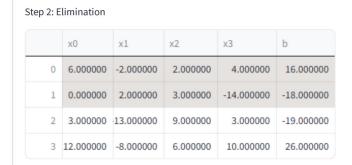
Select step



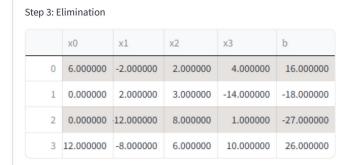
1



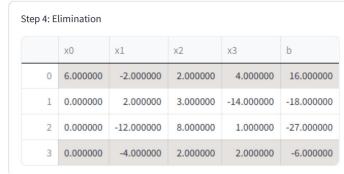
Column 0: pivot row 2 selected with scaled ratio 1.000. Swapped row 0 and 2.



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

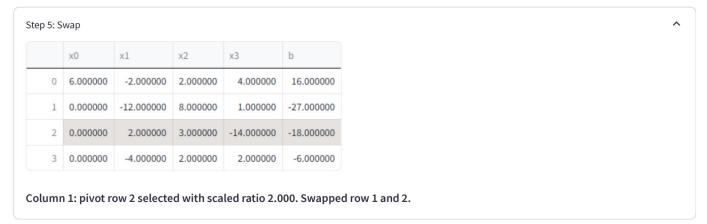


Row 2: eliminate A[2,0] using multiplier 3.0/6.0 = 0.500.

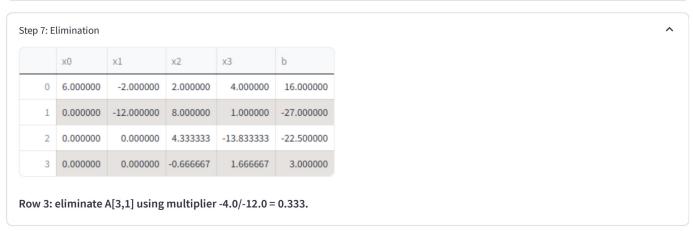


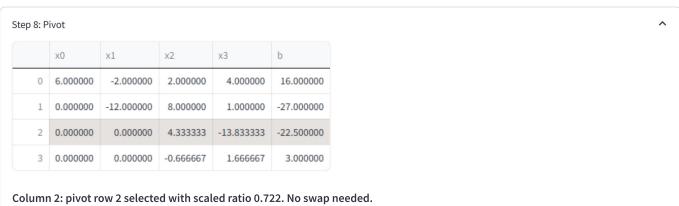
localhost:8501 5/14

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

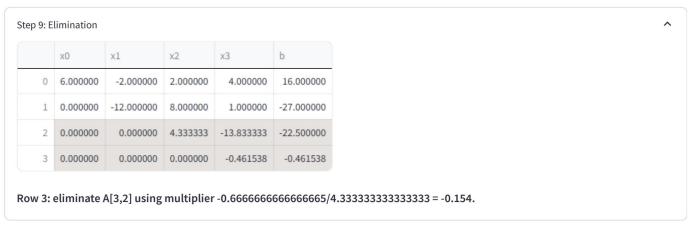


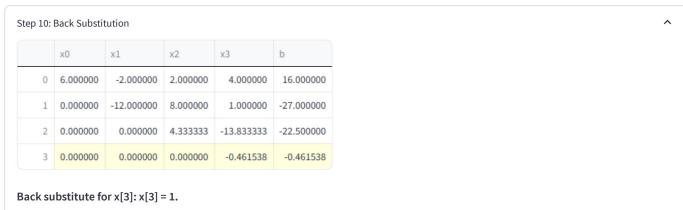
Step 6: Elimination x0 x1 x2 хЗ 6.000000 -2.000000 2.000000 4.000000 16.000000 1 0.000000 -12.000000 8.000000 1.000000 -27.000000 2 0.000000 0.000000 4.333333 -13.833333 -22.500000 0.000000 -4.000000 2.000000 2.000000 -6.000000 Row 2: eliminate A[2,1] using multiplier 2.0/-12.0 = -0.167.



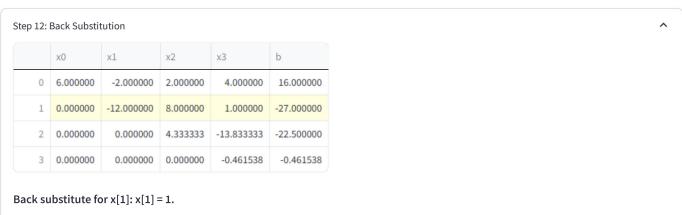


localhost:8501 6/14









Step 13: Back Substitution

localhost:8501 7/14

	x0	x1	x2	х3	b
0	6.000000	-2.000000	2.000000	4.000000	16.000000
1	0.000000	-12.000000	8.000000	1.000000	-27.000000
2	0.000000	0.000000	4.333333	-13.833333	-22.500000
3	0.000000	0.000000	0.000000	-0.461538	-0.461538

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

### Solution



### **Solution Verification**

Residual (Ax - b): [0.00000000e+00 0.00000000e+00 3.55271368e-15 3.55271368e-15]

Infinity Norm of Residual: 3.553e-15

## **Report Preview**

# **Gaussian Elimination Report**

#### John Akujobi

MATH 374: Scientific Computation (Spring 2025), South Dakota State University Professor: Dr Kimn, Dept. of Math & Statistics

GitHub: <u>jakujobi</u>

## **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.]
```

localhost:8501 8/14

## **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

```
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
       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.")
        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
```

localhost:8501 9/14

```
ratio = float(ratios[idx_max]) # scaled ratio for pivot
    # Check for near-singular matrix
    if abs(A[p, k]) < tol:</pre>
        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:</pre>
        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()
```

localhost:8501 10/14

```
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]
```

localhost:8501 11/14

```
[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.

### **Step 7: Elimination**

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

### Step 8: Pivot

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

## **Step 9: Elimination**

Row 3: eliminate A[3,2] using multiplier -0.6666666666666666/4.333333333333333 = -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, 4.33333333333333, -13.833333333334, -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, -13.8333333333334, -22.5]

[0.0, 0.0, 0.0, -0.46153846153846145, -0.46153846153846123]
```

## **Step 11: Back Substitution**

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

localhost:8501 12/14

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

[0.0, -12.0, 8.0, 1.0, -27.0]

[0.0, 0.0, 4.3333333333333, -13.83333333333334, -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, 4.3333333333333, -13.83333333333333, -22.5]

[0.0, 0.0, 0.0, -0.46153846153846145, -0.46153846123]
```

### **Step 13: Back Substitution**

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

### Solution

```
(3.00000000000000, 0.999999999999, -2.00000000000013, 0.99999999999)
```

## **Performance Metrics**

Execution Time: 0.000625 seconds Estimated Floating-point Operations: 42

## **Solution Verification**

Residual (Ax - b): [0.00000000e+00 0.0000000e+00 3.55271368e-15 3.55271368e-15] Infinity Norm of Residual: 3.553e-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.

Download report (Markdown)

PDF export disabled: install WeasyPrint or FPDF to enable this feature.

localhost:8501 13/14

## **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.

localhost:8501 14/14