# **Gaussian Elimination with Scaled Partial Pivoting**

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

# **3×3 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. 1. 2.]
[1. 2. 0.]
[0. 1. 1.]]
```

```
[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 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	1	p <sup>l</sup> vot_row	mu <sup>ltī</sup> p <sup>li</sup> er	ra <sup>ti</sup> o	value
0	1	pivot	0		0		1.0	

localhost:8501

	Step#	Туре	k	i	pivot_row	multiplier	ratio	value
1	2	elimination	0	1		0.333333333333333		
2	3	elimination	0	2		0.0		
3	4	swap	1		2		1.0	
4	5	elimination	1	2		1.6666666666666667		
5	6	back_substitution		2				0.1428571428571427
6	7	back_substitution		1				2.857142857142857
7	8	back_substitution		0				2.285714285714286

#### **Intermediate Matrices**

#### Original Matrix (k=0) After Pivot (k=0)

	x0	x1	x2
0	3	1	2
1	1	2	0
2	0	1	1

	x0	x1	x2
0	3	1	2
1	1	2	0
2	0	1	1

## After Elimination (k=0)

	х0	x1	x2
0	3	1	2
1	0	1.6667	-0.6667
2	0	1	1

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

	x0	x1	x2
0	3	1	2
1	1	2	0
2	0	1	1

	x0	x1	x2
0	3	1	2
1	0	1	1
2	0	1.6667	-0.6667

# After Elimination (k=1)

	x0	x1		x2
0	3		1	2
1	0	r.	1	1
2	0		0	-2.3333

#### **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
    _____
   x : ndarray
       Solution vector
```

2/11 localhost:8501

```
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], :]
        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]
```

localhost:8501 3/11

```
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**

Execution Time: 0.000428 seconds

Estimated Floating-point Operations: 18

Select step



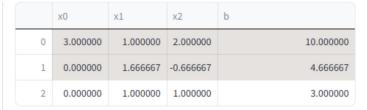
8



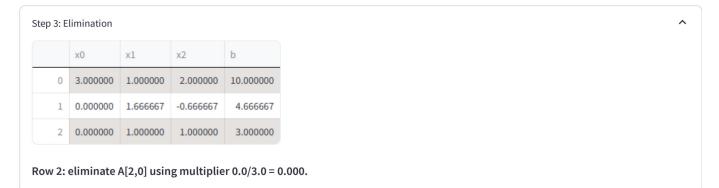
Column 0: pivot row 0 selected with scaled ratio 1.000. No swap needed.

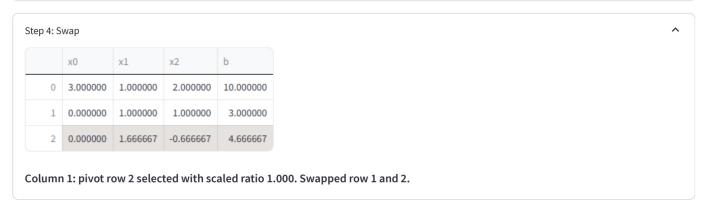
Step 2: Elimination

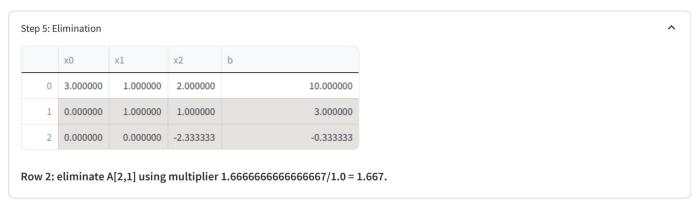
localhost:8501 4/11



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









localhost:8501 5/11





#### Solution

2.2857 2.8571 0.1429

#### **Solution Verification**

Residual (Ax - b): [0. 0. 0.]

Infinity Norm of Residual: 0.000e+00

## **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: jakujobi

# **Problem Statement**

Matrix A:

localhost:8501 6/11

```
[[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

```
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)
```

localhost:8501 7/11

```
# 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], :]
        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)
```

localhost:8501 8/11

```
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</pre>
```

# **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.

# **Step 3: Elimination**

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

# 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, 1.6666666666666666, -0.6666666666666, 4.666666666666]
```

localhost:8501 9/11

#### Step 5: Elimination

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

#### **Step 6: Back Substitution**

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

#### **Step 7: Back Substitution**

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

## **Step 8: Back Substitution**

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

# **Solution**

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

# **Performance Metrics**

Execution Time: 0.000338 seconds Estimated Floating-point Operations: 18  $\,$ 

# **Solution Verification**

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

## **References & Notes**

• Gaussian elimination – Wikipedia

localhost:8501 10/11

- 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.

#### **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 11/11