Numerical Matrix Analysis

Notes #16 — Systems of Equations
Gaussian Elimination / LU-Factorization with Pivoting

Peter Blomgren
⟨blomgren@sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Spring 2024
(Revised: March 19, 2024)

Outline

1. Student Learning Targets, and Objectives
   - SLOs: Gaussian Elimination & LU-Factorization
2. Gaussian Elimination
   - Introduction: GE — Something Familiar
   - GE, Backward Substitution, and LU-Factorization
   - Computational Complexity
3. GE: Instabilities, and Improvements
   - Partial Pivoting
   - Scaled Partial Pivoting
   - Complete Pivoting

SAN DIEGO STATE UNIVERSITY

Student Learning Targets, and Objectives

Target Gaussian Elimination

- **Objective** The three fundamental row-reduction operations
- **Objective** Know how the L and U factors arise from Gaussian Elimination (to ~~Reduced~~ Row Echelon Form)
- **Objective** Stability issues, and potential remedies: Pivoting Strategies

San Diego State University

Gaussian Elimination: Introduction

We look at a familiar algorithm — Gaussian Elimination.

- The "pure" form.
- Connection to LU-factorization.
- Pivoting strategies to improve stability:
  - Scaled Partial Pivoting
  - (Rescaled) Scaled Partial Pivoting
  - Complete Pivoting

The Augmented Matrix [A b]

Given a matrix A and a column vector $\vec{b}$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix},$$

we define the **augmented matrix**

$$[A \;\; \vec{b}] = \left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right]$$

We are going to operate on this augmented matrix using 3 fundamental operations...

SAN DIEGO STATE UNIVERSITY

Three Basic Operations on the Linear System / Augmented Matrix

We use three operations to simplify a linear system:

op#1 **Scaling** — Equation#i (E_i) can be multiplied by any non-zero constant λ with the resulting equation used in place of E_i. We denote this operation $(\mathbf{E_i}) \leftarrow (\lambda \mathbf{E_i})$.

op#2 **Scaled Addition** — Equation#j (E_j) can be multiplied by any non-zero constant λ and added to Equation#i (E_i) with the resulting equation used in place of E_i. We denote this operation $(\mathbf{E_i}) \leftarrow (\mathbf{E_i} + \lambda \mathbf{E_j})$.

op#3 **Reordering** — Equation#j (E_j) and Equation#i (E_i) can be transposed in order. We denote this operation $(\mathbf{E_i}) \leftrightarrow (\mathbf{E_j})$.

Gaussian Elimination, Backward Substitution, and LU-Factorization

The goal is to apply a sequence of the operations on the augmented matrix

$$[A\ \ \vec{b}] = \left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right],$$

in order to transform it into the **upper triangular form**

$$\left[\begin{array}{ccc|c} \tilde{a}_{11} & \tilde{a}_{12} & \tilde{a}_{13} & \tilde{b}_1 \\ 0 & \tilde{a}_{22} & \tilde{a}_{23} & \tilde{b}_2 \\ 0 & 0 & \tilde{a}_{33} & \tilde{b}_3 \end{array}\right].$$

From this form we use **backward substitution** to get the solution:

$$x_3 \leftarrow \tilde{b}_3/\tilde{a}_{33}, \quad x_2 \leftarrow (\tilde{b}_2 - \tilde{a}_{23}x_3)/\tilde{a}_{22},$$

$$x_1 \leftarrow (\tilde{b}_1 - \tilde{a}_{12}x_2 - \tilde{a}_{13}x_3)/\tilde{a}_{11}.$$

GE+BS+LU

Given an augmented matrix

$$C = [A \;\; \vec{b}] = \left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \dots & a_{1m} & b_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} & b_2 \\ a_{31} & a_{32} & a_{33} & \dots & a_{3m} & b_3 \\ \vdots & & & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mm} & b_m \end{array}\right]$$

We first make all the sub-diagonal entries in the first column zero:

for j = 2:m [Eliminate the first column]
  $\ell_{j1} \leftarrow -c_{j1}/c_{11}$
  $r_j \leftarrow (\ell_{j1} r_1 + r_j)$ [r_j denotes elements in the jth row]
end

SAN DIEGO STATE UNIVERSITY

The pattern is clear... For a full implementation we eliminate all the sub-diagonal elements in columns $1 \rightarrow (m-1)$:

```
for i = 1:(m-1)
  for j = (i+1):m                  [Eliminate the i th column]
    ℓ_ji  ← −c_ji / c_ii
    r_j   ← (ℓ_ji r_i + r_j)      [r_j -- elements in the j th row]
  end
end
```

SAN DIEGO STATE UNIVERSITY

GE+BS+LU

After the elimination step, we have the following scenario — the augmented matrix is now upper triangular; we identify the upper triangular part U, and the modified right-hand-side $\tilde{b}$, and collect the multipliers in matrices M_j

$$\tilde{\mathtt{C}} = [\mathtt{U}\ \tilde{b}] = \left[\begin{array}{ccccc|c} u_{11} & u_{12} & u_{13} & \cdots & u_{1m} & \tilde{b}_1 \\ & u_{22} & u_{23} & \cdots & u_{2m} & \tilde{b}_2 \\ & & u_{33} & \cdots & u_{3m} & \tilde{b}_3 \\ & & & \ddots & \vdots & \vdots \\ & & & & u_{mm} & \tilde{b}_m \end{array}\right], \quad \mathtt{M}_1 = \begin{bmatrix} 1 & & & & \\ \ell_{21} & 1 & & & \\ \ell_{31} & 0 & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{m1} & 0 & \cdots & 0 & 1 \end{bmatrix}$$

We have the relation

$$\underbrace{M_{m-1} \cdot M_{m-2} \cdots M_1}\cdot C = M \cdot C = M \cdot [A \mid \vec{b}] = [U \mid \tilde{b}] = \tilde{C}$$

GE+BS+LU

Now, if we are looking for the solution to $A\vec{x} = \vec{b}$, we simply apply backward substitution to the $[U \,|\, \tilde{b}]$ system.

If we define $L = M^{-1}$; — think of it as inverting (undoing) the triangularization of A

$$\mathrm{L} = M_1^{-1} M_2^{-1} \cdots M_{m-1}^{-1} = \begin{bmatrix} 1 & & & & \\ -\ell_{21} & 1 & & & \\ -\ell_{31} & -\ell_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ -\ell_{m1} & -\ell_{m2} & \cdots & -\ell_{m,m-1} & 1 \end{bmatrix}$$

Then we have the **LU-Factorization** of A

$$A = LU.$$

Gaussian Elimination $\leftrightarrow$ Matrix Multiplications — Supplemental

We can view the entire GE-algorithm as a sequence of matrix multiplications:

$$\underbrace{M_{m-1}M_{m-2}\cdots M_2 M_1}_{M} A = U$$

and it follows that we can write

$$A = M^{-1}U = [M_1]^{-1}[M_2]^{-1}\ldots[M_{m-2}]^{-1}[M_{m-1}]^{-1}U$$

The multiplication by the matrices $[M_j]$ correspond to scaled row-addition; the inverse operation is scaled row-subtraction, hence

$$[M_j]^{-1} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -\ell_{j+1,j} & 1 & & \\ & & \vdots & & \ddots & \\ & & -\ell_{m,j} & & & 1 \end{bmatrix}$$

Next, we check this!

Checking the Inverses of M_j — Supplemental

$$[M_j]^{-1}[M_j] = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -\ell_{\mathbf{j+1,j}} & \mathbf{1} & & \\ & & \vdots & & \ddots & \\ & & -\ell_{\mathbf{m,j}} & & & \mathbf{1} \end{bmatrix} \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & \mathbf{1} & & & \\ & & \ell_{\mathbf{j+1,j}} & 1 & & \\ & & \vdots & & \ddots & \\ & & \ell_{\mathbf{m,j}} & & & 1 \end{bmatrix}$$

When we perform the matrix-matrix multiplication, the sub-diagonal elements of $[M_j]^{-1}$ (in column j, row $k \geq j$) will multiply elements in row j (column k) of $[M_j]$ (only the **1** on the diagonal). When that happens, the diagonal k-k element of $[M_j]^{-1}$ will multiply the k-j-element of $[M_j]$, and we get

$$\text{Product}(k,j) = -\ell_{k,j} \cdot 1 + 1 \cdot \ell_{k,j} = 0, \;\; k > j$$

All other off-diagonal elements are formed by (something) multiplying zero.

In summary, the only non-zeros elements in the product are the diagonal elements, which are all 1.

In the same way $[M_j][M_j]^{-1} = I_n$, hence the matrix we denoted $[M_j]^{-1}$ really is the inverse of $[M_j]$.

Nailing Down the L in $A = L \cdot U$ — Supplemental

We now have expression for all the $[M_j]^{-1}$-matrices in the product $M^{-1} = [M_1]^{-1}[M_2]^{-1} \dots [M_{m-2}]^{-1}[M_{m-1}]^{-1}$. Consider $[M_1]^{-1}[M_2]^{-1}$:

$$\left[\begin{array}{c|cccc} 1 & & & & \\ \hline -\ell_{2,1} & 1 & & & \\ -\ell_{3,1} & & 1 & & \\ \vdots & & & \ddots & \\ -\ell_{m,1} & & & & 1 \end{array}\right] \left[\begin{array}{c|cccc} 1 & & & & \\ \hline & 1 & & & \\ & -\ell_{3,2} & 1 & & \\ & \vdots & & \ddots & \\ & -\ell_{m,2} & & & 1 \end{array}\right] = \begin{bmatrix} 1 & & & & \\ -\ell_{2,1} & 1 & & & \\ -\ell_{3,1} & -\ell_{3,2} & 1 & & \\ \vdots & \vdots & & \ddots & \\ -\ell_{m,1} & -\ell_{m,2} & & & 1 \end{bmatrix}$$

The argument can be extended to the entire product to show that

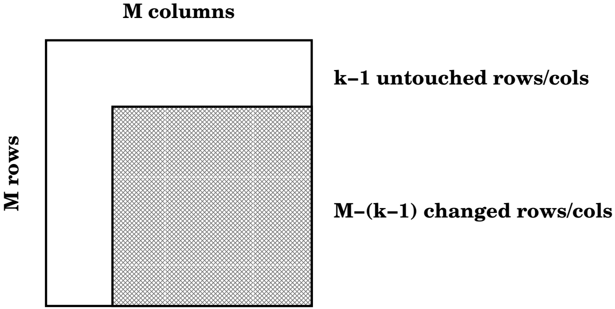
$$L = M^{-1} = \begin{bmatrix} 1 & & & & \\ -\ell_{2,1} & 1 & & & \\ -\ell_{3,1} & -\ell_{3,2} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ -\ell_{m,1} & -\ell_{m,2} & \cdots & -\ell_{m,m-1} & 1 \end{bmatrix}$$

Which is the matrix we build in our LU-factorization core.

SAN DIEGO STATE UNIVERSITY

GE+BS: Work Required

Gaussian Elimination: Consider the k th elimination step:



In this step we need to touch (read from cache/memory, apply addition and/or multiplication) the shaded elements. The work required is directly proportional to the number shaded elements i^2, where $i = (M - (k-1))$.

GE+BS: Work Required

We have $(M-1)$ elimination steps where k runs from 1 to $(M-1)$, hence i runs from M down to 2. The total work is

$$\sum_{i=2}^{M} 2i^2 = \frac{M(M+1)(2M+1)}{3} - 1 = \mathcal{O}\left(\frac{2M^3}{3}\right).$$

Solving $A\vec{x} = \vec{b}$ by factorization — work comparison for the factorization step $(m = n)$:

GE+BS: Work Required

LU-Factorization	$\frac{2m^3}{3}$
QR: Householder, "Q-less"	$\frac{4m^3}{3}$
QR: Gram-Schmidt	$2m^3$
SVD	$13m^3$

↓ more work ↓

↓ more stability* ↓

* GS-QR is not necessarily more stable than H-QR...

Instability of Gaussian Elimination / LU-Factorization

As described, GE/LU **can run into stability issues** — consider the multipliers in the light of stability and floating-point errors

$$\tilde{\ell}_{ji} = -c_{ij} \oslash c_{ii} = -\frac{c_{ij}}{c_{ii}}(1+\epsilon), \ \ |\epsilon| \leq \varepsilon_{\text{mach}}$$

Hence, the absolute errors introduced in the multipliers are

$$\delta\ell_{ji} \sim \varepsilon_{\text{mach}} \left(\frac{c_{ij}}{c_{ii}} \right)$$

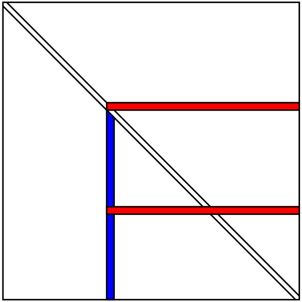
and if c_{ii} is close to zero, then the error may be very large (especially in comparison with other entries in the matrix).

We need to fix this...

Clearly, **the smaller the multipliers, the smaller the errors...**

Pivoting Strategies — Partial Pivoting

It is fairly easy to re-arrange the computation so that all multipliers are bounded by 1.



Partial pivoting adds $\frac{m^2}{2}$ comparisons to the algorithm.

Figure: Illustration of elimination on the k th level. We search for the largest (in magnitude) pivot element in the k th column, among the diagonal+sub-diagonal elements (vertical blue band). Then we interchange the k th row with the row with the maximal pivot (illustrated with two horizontal red bands).

Gaussian Elimination with Partial Pivoting $U = [A\ \vec{b}]$

```
L = eye(m);  P=eye(m);  U = [A b];
for k = 1:(m-1)
  Umax              = max( abs(U(k:m,k)) );
  Umax_index        = find( abs(U(k:m,k)) == Umax );
  j                 = Umax_index(1) + (k-1);
  U([j k],k:(m+1)) = U([k j],k:(m+1));
  L([j k],1:(k-1)) = L([k j],1:(k-1));
  P([j k],:)        = P([k j],:);
  for j=(k+1):m
    L(j,k)        =  U(j,k) / U(k,k);
    U(j,k:(m+1)) =  U(j,k:m+1) - L(j,k)*U(k,k:(m+1));
  end
end
```

The algorithm yields

$$\mathbf{PA} = \mathbf{LU}.$$

It is much more stable than our initial two implementations of Gaussian Elimination, but it is **not** fail-safe.

Row- and Column-Swapping in Python

```
# Swap Rows r1 and r2
A = np.array([[...], ..., [...]])
A[[r1, r2]] = A[[r2, r1]]
```

```
# Swap Columns c1 and c2
A = np.array([[...], ..., [...]])
A[:, [c1, c2]] = A[:, [c2, c1]]
```

Gaussian Elimination with Partial Pivoting: Breakdown

If we apply GE+PP to a system where the **scales** of the different equations are significantly different, the algorithm may break down (unnecessarily lose precision) *e.g*

$$\begin{bmatrix} 1 & -2 & 3 \\ 1,000,000 & 2,000,000 & 3,000,000 \\ 0.000001 & -0.000002 & -0.000003 \end{bmatrix} \vec{x} = \begin{bmatrix} 4 \\ 5,000,000 \\ 0.000001 \end{bmatrix}$$

In order to improve stability of GE+PP we must take scale into consideration.

One definition of scale: `s(i) = max(abs(B(i,:)))`, *i.e.* the scale of row #i equals to the magnitude of the largest element on that row.

Gaussian Elimination with Scaled Partial Pivoting

"Scale Invariant PP" ???

We can pre-compute the scales `s(i)` and make the pivoting decision based on the values of `B(i,i)/s(i)` and `B(j,i)/s(j)`, `j=(i+1):n`.

```
s = zeros(m,1);
for i=1:m
  s(i) = max(abs(B(i,:)));
end
for i=1:(m-1)
  Bmax = max(abs(B(i:m,i)./s(i:m)));
  Bmax_index = find( abs(B(i:m,i)./s(i:m)) == Bmax );
  j = Bmax_index(1) + (i-1);
  B([j i],i:(m+1)) = B([i j],i:(m+1));
  L([j i],1:(i-1)) = L([i j],1:(i-1));
  P([j i],:)  = P([i j],:);
  for j=(i+1):m
    L(j,i) = -B(j,i) / B(i,i);
    B(j,i:(m+1)) = L(j,i)*B(i,i:(m+1)) + B(j,i:(m+1));
  end
end
```

GE+SPP: Work Comparison

```
s = zeros(m,1);
for i=1:m
  s(i) = max(abs(B(i,:)));
end
for i=1:(m-1)
  Bmax = max(abs(B(i:m,i)./s(i:m)));
  Bmax_index = find( abs(B(i:m,i)./s(i:m)) == Bmax );
  j = Bmax_index(1) + (i-1);
  B([j i],i:(m+1)) = B([i j],i:(m+1));
  L([j i],1:(i-1)) = L([i j],1:(i-1));
  P([j i],:)  = P([i j],:);
  for j=(i+1):m
    L(j,i) = -B(j,i) / B(i,i);
    B(j,i:(m+1)) = L(j,i)*B(i,i:(m+1)) + B(j,i:(m+1));
  end
end
```

Note that the scale computation touches every element in the matrix, hence it adds

$$\mathcal{O}\left(\mathbf{m}^2\right) \textbf{ additional operations.}$$

Since this algorithm overall requires $\mathcal{O}\left(m^3\right)$ operations, the overhead of scaled partial pivoting does not add a significant amount of work.

GE+SPP: Wait a Minute! — The Scale Changes

Since we are modifying the rows in each elimination step, it seems likely that the scale of the row change. Should we recompute them???

```
s = zeros(m,1);
for i=1:(m-1)
  for k=i:m
    s(k) = max(abs(B(k,:)));
  end
  Bmax = max(abs(B(i:m,i)./s(i:m)));
  Bmax_index = find( abs(B(i:m,i)./s(i:m)) == Bmax );
  j = Bmax_index(1) + (i-1);
  B([j i],i:(m+1)) = B([i j],i:(m+1));
  L([j i],1:(i-1)) = L([i j],1:(i-1));
  P([j i],:)  = P([i j],:);
  for j=(i+1):m
    L(j,i) = -B(j,i) / B(i,i);
    B(j,i:(m+1)) = L(j,i)*B(i,i:(m+1)) + B(j,i:(m+1));
  end
end
```

Let's call this GE+Rescaled-SPP (GE+RSPP). Since we are touching all the remaining elements in the matrix in each iteration, this configuration adds

$$\mathcal{O}\left(m^3\right) \text{ additional operations,}$$

which is a significant amount of work.

GE with Complete Pivoting

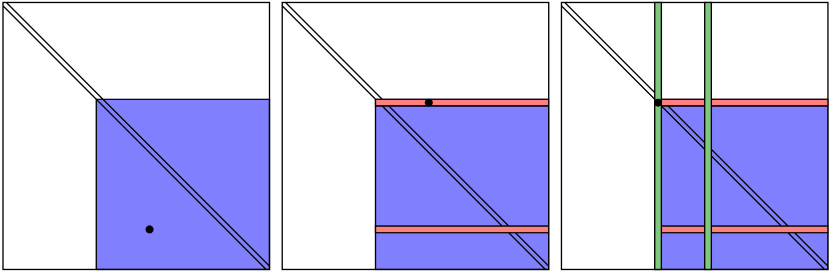
GE+CP

If/when a problem warrants this (GE+RSPP) approach due to high accuracy demands, and we are willing to trade significant time/work for it) **complete pivoting** should be used instead.

```
for i=1:(m-1)
  Bmax = max(max(abs(B(i:m,i:m))));
  [Bmax_r,Bmax_c] = find( abs(B(i:m,i:m)) == Bmax );
  j_r = Bmax_r(1) + (i-1);
  j_c = Bmax_c(1) + (i-1);
  B([j_r i],i:(m+1)) = B([i j_r],i:(m+1));
  L([j_r i],1:(i-1)) = L([i j_r],1:(i-1));
  P([j_r i],:)       = P([i j_r],:);
  B(:,[j_c i])       = B(:,[i j_c]);
  for j=(i+1):m
   L(j,i) = -B(j,i) / B(i,i);
   B(j,i:(m+1)) = L(j,i)*B(i,i:(m+1)) + B(j,i:(m+1));
  end
end
```

WARNING!!! — When the columns are interchanged, the unknowns are re-ordered. We have to implement extra book-keeping in order to keep track!

Illustration: Gaussian Elimination with Complete Pivoting



[Left] Illustration of elimination on the kth level. We search for the largest (in magnitude) pivot element in the sub-matrix indicated with blue; the pivot is marked with a black dot.

[Center] We interchange the corresponding rows, to move the pivot to the "active" row.

[Right] We interchange the columns to move the pivot to the "active" A_{kk} pivot location.

GE with Complete Pivoting Book-keeping GE+CP

```
col_idx = (1:m)';
for i=1:(m-1)
  Bmax = max(max(abs(B(i:m,i:m))));
  [Bmax_r,Bmax_c] = find( abs(B(i:m,i:m)) == Bmax );
  j_r = Bmax_r(1) + (i-1);
  j_c = Bmax_c(1) + (i-1);
  B([j_r i],i:(m+1)) = B([i j_r],i:(m+1));
  L([j_r i],1:(i-1)) = L([i j_r],1:(i-1));
  P([j_r i],:)       = P([i j_r],:);
  B(:,[j_c i])       = B(:,[i j_c]);
  col_idx([j_c i])   = col_idx([i j_c]);
  for j=(i+1):m
   L(j,i) = -B(j,i) / B(i,i);
   B(j,i:(m+1)) = L(j,i)*B(i,i:(m+1)) + B(j,i:(m+1));
  end
end
```

After completion, `col_idx(i)` contains the original index of the variable currently called `x(i)`.

After GE+CP, we solve for $\vec{x}$ using standard Backward Substitution, then we use the `col_idx` array to put the solution array back in the correct order:

GE with Complete Pivoting — Reconstitution — GE+CP

GE+CP+BS gives us a vector with the order of the x_i's "scrambled" from the column interchanges. To unscramble:

```
I  = eye(n);
P2 = I(:,col_idx);
x  = P2*x;
```

and we have solved $A\vec{x} = \vec{b}$ in the most stable way! (In the framework of Gaussian elimination, that is...)

Note: We can handle the row-pivoting in the same way (using an "index-array") `row_idx`.

Next Time

- A formal look at stability of Gaussian Elimination.
- Gaussian Elimination for **Hermitian Positive Definite Matrices:**
  - Cholesky Factorization.

Homework (Not Explicitly Due...)

Read Trefethen & Bau's take on Gaussian Elimination and Pivoting, pp. 147–162.