Gaussian Elimination / LU-Factorization with Pivoting

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

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 2016
Outline

1. Gaussian Elimination
   - Introduction: GE — Something Familiar
   - GE, Backward Substitution, and LU-Factorization
   - Computational Complexity

2. GE: Instabilities, and Improvements
   - Partial Pivoting
   - Scaled Partial Pivoting
   - Complete Pivoting
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 \(\bar{b}\)

\[
A = \begin{bmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]
\[
\bar{b} = \begin{bmatrix}
b_1 \\
b_2 \\
b_3
\end{bmatrix}
\]

We can define the **augmented matrix**

\[
[A \ b] = \begin{bmatrix}
a_{11} & a_{12} & a_{13} & b_1 \\
a_{21} & a_{22} & a_{23} & b_2 \\
a_{31} & a_{32} & a_{33} & b_3
\end{bmatrix}
\]

We are going to operate on this augmented matrix using 3 basic operations...
We use three operations to simplify a linear system:

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

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

**op#3 Reordering** — Equation\#j \((E_j)\) and Equation\#i \((E_i)\) can be transposed in order. We denote this operation \((E_i) \leftrightarrow (E_j)\).
Gaussian Elimination, Backward Substitution, and LU-Factorization

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

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

in order to transform it into the upper triangular form

\[
\begin{bmatrix}
\tilde{a}_{11} & \tilde{a}_{12} & \tilde{a}_{13} & \tilde{b}_1 \\
0 & \tilde{a}_{22} & \tilde{a}_{23} & \tilde{b}_2 \\
0 & 0 & \tilde{a}_{33} & \tilde{b}_3 \\
\end{bmatrix}.
\]

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

\[
x_3 = \frac{\tilde{b}_3}{\tilde{a}_{33}}, \quad x_2 = \frac{\tilde{b}_2 - \tilde{a}_{23}x_3}{\tilde{a}_{22}},
\]

\[
x_1 = \frac{\tilde{b}_1 - \tilde{a}_{12}x_2 - \tilde{a}_{13}x_3}{\tilde{a}_{11}}.
\]
Given an augmented matrix

\[
C = [A \ b] = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} & \ldots & a_{1m} & b_1 \\
  a_{21} & a_{22} & a_{23} & \ldots & a_{2m} & b_2 \\
  a_{31} & a_{32} & a_{33} & \ldots & a_{3m} & b_3 \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
  a_{m1} & a_{m2} & a_{m3} & \ldots & a_{mm} & b_m
\end{bmatrix}
\]

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

for \( j=2:m \)  \[\text{[Eliminate the first column]}\]

\[
l_{j1} = -\frac{c_{j1}}{c_{11}}
]\]

\[(l_{j1}r_1 + r_j) \rightarrow (r_j) \quad \text{[} r_j \text{ denotes elements in the } j\text{th row]}\]

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

\[
\text{for } i=1:(m-1) \\
\quad \text{for } j=(i+1):m \quad [\text{Eliminate the } i\text{th column}] \\
\quad \quad l_{ji} = -c_{ji}/c_{ii} \\
\quad \quad (l_{ji}r_i + r_j) \rightarrow (r_j) \quad [r_j \quad \text{-- elements in the } j\text{th row}] \\
\quad \text{end} \\
\text{end}
\]
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 $M_j$

$$\tilde{c} = [U \, \tilde{b}] = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1m} & \tilde{b}_1 \\ u_{22} & u_{23} & \cdots & u_{2m} & \tilde{b}_2 \\ u_{33} & \cdots & u_{3m} & \tilde{b}_3 \\ \vdots & \vdots & \vdots & \vdots \\ u_{mm} & \tilde{b}_m \end{bmatrix}, \ M_1 = \begin{bmatrix} 1 & 1 \\ l_{21} & 1 \\ l_{31} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ l_{m1} & 0 & \cdots & 0 & 1 \end{bmatrix}$$

We have the relation

$$M_{m-1} \cdot M_{m-2} \cdots M_1 \cdot C = M \cdot C = M \cdot [A \mid \tilde{b}] = [U \mid \tilde{b}] = \tilde{C}$$
Now, if we are looking for the solution to $A\bar{x} = \bar{b}$, 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 $A$

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

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

$$A = LU.$$
We can view the entire GE-algorithm as a sequence of matrix multiplications:

\[
M_{m-1}M_{m-2} \cdots M_2 M_1 A = U
\]

and it follows that we can write

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

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

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

We must check this!
Checking the Inverses of $M_j$

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

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 $k\cdot k$ element of $[M_j]^{-1}$ will multiply the $k\cdot j$-element of $[M_j]$, and we get

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

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 $[M_j]$. 

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩

GE / LU-Factorization with Pivoting — (12/29)
Nailing Down the $L$ in $A = L \cdot U$

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

$$
\begin{bmatrix}
1 & 1 & \cdots & 1 \\
-l_{2,1} & 1 & \cdots & 1 \\
-l_{3,1} & \vdots & \ddots & \vdots \\
-l_{m,1} & \vdots & \ddots & 1
\end{bmatrix}
\begin{bmatrix}
1 & 1 & \cdots & 1 \\
-l_{3,2} & 1 & \cdots & 1 \\
-l_{m,2} & \vdots & \ddots & 1 \\
-l_{m,1} & \vdots & \ddots & 1
\end{bmatrix} =
\begin{bmatrix}
1 & -l_{2,1} & 1 & \cdots \\
-l_{3,1} & -l_{3,2} & 1 & \cdots \\
-l_{m,1} & -l_{m,2} & \cdots & 1
\end{bmatrix}
$$

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

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

Which is the matrix we build in our LU-factorization core.
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 $i = (N - (k - 1))$. 

M columns

k−1 untouched rows/cols

M−(k−1) changed rows/cols
We have \((N - 1)\) elimination steps where \(k\) runs from 1 to \((N - 1)\), hence \(i\) runs from \(N\) down to 2. The total work is proportional to

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

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

\[ \frac{2m^3}{3} \]

## QR: Householder, “Q-less”

\[ \frac{4m^3}{3} \]

## QR: Gram-Schmidt

\[ 2m^3 \]

## SVD

\[ 13m^3 \]
Instability of Gaussian Elimination / LU-Factorization

As described, GE/LU is not stable — consider the multipliers in the light of stability and floating-point errors

\[ \hat{l}_{ji} = -c_{ij} \bigotimes c_{ii} = -\frac{c_{ij}}{c_{ii}} (1 + \epsilon), \quad |\epsilon| \leq \epsilon_{\text{mach}} \]

Hence, the absolute errors introduced in the multipliers are

\[ \delta l_{ji} \sim \epsilon_{\text{mach}} \left( \frac{c_{ij}}{c_{ii}} \right) \]

and if \( c_{ij} \) is close to zero, then the error may be very large.

We need to fix this...

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

\[ C = [A \bar{b}] \]

\[
L = \text{eye}(m); \quad P = \text{eye}(m); \\
\text{for } i=1:(m-1) \\
\quad C_{\text{max}} = \max(\text{abs}(C(i:m,i)));
\]
\[
\text{Cmax_index} = \text{find}( \text{abs}(C(i:m,i)) == \text{Cmax });
\]
\[
\quad j = \text{Cmax_index}(1) + (i-1);
\]
\[
C([j i],i:(m+1)) = C([i j],i:(m+1));
\]
\[
L([j i],1:(i-1)) = L([i j],1:(i-1));
\]
\[
P([j i],:) = P([i j],:);
\]
\[
\text{for } j=(i+1):m \\
\quad L(j,i) = -C(j,i) / C(i,i);
\]
\[
C(j,i:(m+1)) = L(j,i) * C(i,i:(m+1)) + C(j,i:(m+1));
\]
end
end

The algorithm yields

\[ PA = LU. \]

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

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

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

One definition of scale: \( s(i) = \max(\text{abs}(B(i,:))) \), i.e. the scale of row \( \neq i \) equals to the magnitude of the largest element on that row.
Gaussian Elimination with Scaled Partial Pivoting

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

```matlab
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));
    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
```

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩
GE+SPP: Work Comparison

```matlab
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));
    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

$$O\left(m^2\right)$$ additional operations.

Since this algorithm overall requires $$O\left(m^3\right)$$ 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???

\[
s = \text{zeros}(m,1);
\]

\[
\text{for } i=1:(m-1)
\]

\[
\text{for } k=i:m
\]

\[
s(k) = \text{max}(\text{abs}(B(k,:)));
\]

\[
\text{end}
\]

\[
\text{Bmax} = \text{max}(\text{abs}(B(i:m,i)./s(i:m)));
\]

\[
\text{Bmax}_{\text{index}} = \text{find}(\text{abs}(B(i:m,i)./s(i:m)) == \text{Bmax} );
\]

\[
j = \text{Bmax}_{\text{index}}(1) + (i-1);
\]

\[
\text{B}([j \ i],i:(m+1)) = \text{B}([i \ j],i:(m+1));
\]

\[
\text{L}([j \ i],1:(i-1)) = \text{L}([i \ j],1:(i-1));
\]

\[
\text{for } j=(i+1):m
\]

\[
m = -\text{B}(j,i) / \text{B}(i,i);
\]

\[
\text{B}(j,i:(m+1)) = m*\text{B}(i,i:(m+1)) + \text{B}(j,i:(m+1));
\]

\[
\text{end}
\]

\[
\text{end}
\]

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

\[
\mathcal{O}(m^3) \text{ additional operations},
\]

which is a significant amount of work.
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.

```matlab
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));
    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 some book-keeping in order to keep track!
Illustration: Gaussian Elimination with Complete Pivoting

[Left] Illustration of elimination on the $k$th 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.
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)]);
    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 GP+CP, we solve for \( \bar{x} \) using standard Backward Substitution, then we use the col_idx array to put the solution array back in the correct order:
GP+CP+BS gives us a vector with the order of the $x_i$’s “scrambled” from the column interchanges. To unscramble:

$$I = \text{eye}(n);$$
$$P = I(:,\text{col_idx});$$
$$x = P*x;$$

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

— A formal look at stability of Gaussian Elimination.

— Gaussian Elimination for **Hermitian Positive Definite Matrices:**

— Cholesky Factorization.
Read Trefethen & Bau’s take on Gaussian Elimination and Pivoting, pp.147–162.