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
Student Learning Targets, and Objectives

Target  Gaussian Elimination

Objective  Know how the $L$ and $U$ factors arise from Gaussian Elimination (to Reduced Row Echelon Form)

Objective  ...

Objective  ...

SLOs: Gaussian Elimination & LU-Factorization
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 $\vec{b}$

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

we define the **augmented matrix**

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

We are going to operate on this augmented matrix using 3 fundamental 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 $(E_i) \leftarrow (\lambda E_i)$.

**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 $(E_i) \leftarrow (E_i + \lambda E_j)$.

**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:

\[
x_3 \leftarrow \tilde{b}_3 / \tilde{a}_{33}, \quad x_2 \leftarrow (\tilde{b}_2 - \tilde{a}_{23}x_3) / \tilde{a}_{22}, \quad x_1 \leftarrow (\tilde{b}_1 - \tilde{a}_{12}x_2 - \tilde{a}_{13}x_3) / \tilde{a}_{11}.
\]
Given an augmented matrix

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

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

\[ \text{for } j = 2 : m \]

\[ \ell_j \leftarrow -c_{j1} / c_{11} \]

\[ (r_j) \leftarrow (\ell_j r_1 + r_j) \]

[Eliminate the first column]

[r\_ denotes elements in the j\_th row]
The pattern is clear... For a full implementation we eliminate all the sub-diagonal elements in columns 1→(m − 1):

for i=1:(m-1)
  for j=(i+1):m
    \[ \ell_{ji} \leftarrow -c_{ji}/c_{ii} \]
    \[ (r_j) \leftarrow (\ell_{ji}r_i + r_j) \]
  end
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 $M_j$:

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

We have the relation

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

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

Then we have the **LU-Factorization** of \( A \)

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

\[
M_{m-1}M_{m-2} \cdots M_2M_1 A = U
\]

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

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

Next, we check this!
Checking the Inverses of $M_j$

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

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

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

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 $[M_j]$. 

Peter Blomgren (blomgren@sdsu.edu)
We now have expression for all the $[M_j]^{-1}$-matrices in the product $M^{-1} = [M_1]^{-1}[M_2]^{-1} \ldots [M_{m-2}]^{-1}[M_{m-1}]^{-1}$. Consider $[M_1]^{-1}[M_2]^{-1}$:

\[
\begin{bmatrix}
1 & 1 \\
-\ell_{2,1} & 1 \\
-\ell_{3,1} & \ddots \\
-\ell_{m,1} & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
1 & 1 \\
1 & \ddots \\
-\ell_{3,2} & \ddots & 1 \\
-\ell_{m,2} & \cdots & 1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1 \\
-\ell_{2,1} & 1 \\
-\ell_{3,1} & -\ell_{3,2} & 1 \\
\vdots & \vdots & \ddots & \ddots \\
-\ell_{m,1} & -\ell_{m,2} & \cdots & -\ell_{m,m-1} & 1
\end{bmatrix}
\]

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

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

Which is the matrix we build in our LU-factorization core.
**Gaussian Elimination:** Consider the $k$th elimination step:

- **$M$ columns**
- **$k-1$ untouched rows/cols**
- **$M-(k-1)$ changed rows/cols**

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 $i = (M - (k - 1))$. 

**Peter Blomgren** (blomgren@sdsu.edu)
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

<table>
<thead>
<tr>
<th>Method</th>
<th>Computational Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU-Factorization</td>
<td>$\frac{2m^3}{3}$</td>
</tr>
<tr>
<td>QR: Householder, “Q-less”</td>
<td>$\frac{4m^3}{3}$</td>
</tr>
<tr>
<td>QR: Gram-Schmidt</td>
<td>$2m^3$</td>
</tr>
<tr>
<td>SVD</td>
<td>$13m^3$</td>
</tr>
</tbody>
</table>

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

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

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.

We need to fix this...

Clearly, the smaller the multipliers, the smaller the errors...
Pivoting Strategies

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

\[ U = [A \ b] \]

\begin{verbatim}
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
\end{verbatim}

The algorithm yields

\[ PA = LU. \]

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

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

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

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

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));
    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
```

Peter Blomgren ⟨blomgren@sdsu.edu⟩
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));
    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

\[ O(m^2) \] additional operations.

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

```matlab
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

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

which is a significant amount of work.
GE with Complete Pivoting

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));
    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!

Peter Blomgren (blomgren@sdsu.edu)
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));
    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+CP+BS gives us a vector with the order of the $x_i$'s “scrambled” from the column interchanges. To unscramble:

\[
\begin{align*}
I &= \text{eye}(n); \\
P2 &= I(:,\text{col_idx}); \\
x &= P2*x;
\end{align*}
\]

and we have solved $A\vec{x} = \vec{b}$ 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.