Outline

1 Linear Algebra Background

2 Conditioning of linear systems

3 Gauss elimination and \textbf{LU} factorization
   - Pivoting
   - LU factorization
   - Cholesky Factorization
   - Pivoting and Stability

4 Conclusions
The dimension of the column space of a matrix is called the **rank** of the matrix \( A \in \mathbb{R}^{m,n} \),

\[
    r = \text{rank } A \leq \min(m,n).
\]

- If \( r = \min(m, n) \) then the matrix is of **full rank**.
- The **nullspace** \( \text{null}(A) \) or **kernel** \( \text{ker}(A) \) of a matrix \( A \) is the subspace of vectors \( x \) for which

\[
    Ax = 0.
\]

- The dimension of the nullspace is called the **nullity** of the matrix.
- The **orthogonal complement** \( \mathcal{V}^\perp \) or orthogonal subspace of a subspace \( \mathcal{V} \) is the set of all vectors that are orthogonal to every vector in \( \mathcal{V} \).
Fundamental Theorem

- One of the most important theorems in linear algebra: For \( \mathbf{A} \in \mathbb{R}^{m,n} \)

\[
\text{rank } \mathbf{A} + \text{nullity } \mathbf{A} = n.
\]

- In addition to the range and kernel spaces of a matrix, two more important vector subspaces for a given matrix \( \mathbf{A} \) are the:
  - **Row space** or **coimage** of a matrix is the column (image) space of its transpose, \( \text{im } \mathbf{A}^T \).
    
    *Its dimension is also equal to the the rank.*
  - **Left nullspace** or **cokernel** of a matrix is the nullspace or kernel of its transpose, \( \text{ker } \mathbf{A}^T \).

- Second fundamental theorem in linear algebra:

\[
\text{im } \mathbf{A}^T = (\text{ker } \mathbf{A})^\perp
\]

\[
\text{ker } \mathbf{A}^T = (\text{im } \mathbf{A})^\perp
\]
A square matrix $A = [n, n]$ is **invertible or nonsingular** if there exists a **matrix inverse** $A^{-1} = B = [n, n]$ such that:

$$AB = BA = I,$$

where $I$ is the identity matrix (ones along diagonal, all the rest zeros).

The following statements are equivalent for $A \in \mathbb{R}^{n,n}$:

- $A$ is **invertible**.
- $A$ is **full-rank**, $\text{rank } A = n$.
- The columns and also the rows are linearly independent and form a **basis** for $\mathbb{R}^n$.
- The **determinant** is nonzero, $\det A \neq 0$.
- Zero is not an eigenvalue of $A$. 

A. Donev (Courant Institute)
Matrix-matrix multiplication is **not commutative**, $AB \neq BA$ in general. Note $x^Ty$ is a scalar (dot product) so this commutes.

Some useful properties:

\[
C(A + B) = CA + CB \quad \text{and} \quad ABC = (AB)C = A(BC)
\]

\[
(A^T)^T = A \quad \text{and} \quad (AB)^T = B^TA^T
\]

\[
(A^{-1})^{-1} = A \quad \text{and} \quad (AB)^{-1} = B^{-1}A^{-1} \quad \text{and} \quad (A^T)^{-1} = (A^{-1})^T
\]

Instead of **matrix division**, think of multiplication by an inverse:

\[
AB = C \quad \Rightarrow \quad (A^{-1}A)B = A^{-1}C \quad \Rightarrow \quad \begin{cases} 
B = A^{-1}C \\
A = CB^{-1}
\end{cases}
\]
Vector norms

- Norms are the abstraction for the notion of a length or magnitude.
- For a vector \( \mathbf{x} \in \mathbb{R}^n \), the \( p \)-norm is
  \[
  \| \mathbf{x} \|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p}
  \]
  and special cases of interest are:
  - The 1-norm (\( L^1 \) norm or Manhattan distance), \( \| \mathbf{x} \|_1 = \sum_{i=1}^{n} |x_i| \)
  - The 2-norm (\( L^2 \) norm, Euclidian distance),
    \[
    \| \mathbf{x} \|_2 = \sqrt{\mathbf{x} \cdot \mathbf{x}} = \sqrt{\sum_{i=1}^{n} |x_i|^2}
    \]
  - The \( \infty \)-norm (\( L^\infty \) or maximum norm), \( \| \mathbf{x} \|_\infty = \max_{1 \leq i \leq n} |x_i| \)

1. Note that all of these norms are inter-related in a finite-dimensional setting.
Matrix norms

- **Matrix norm induced** by a given vector norm:

  \[
  \|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} \quad \Rightarrow \quad \|Ax\| \leq \|A\| \|x\|
  \]

- The last bound holds for matrices as well, \(\|AB\| \leq \|A\| \|B\|\).

- Special cases of interest are:
  1. The 1-norm or **column sum norm**, \(\|A\|_1 = \max_j \sum_{i=1}^n |a_{ij}|\)
  2. The \(\infty\)-norm or **row sum norm**, \(\|A\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|\)
  3. The 2-norm or **spectral norm**, \(\|A\|_2 = \sigma_1\) (largest singular value)
  4. The Euclidian or **Frobenius norm**, \(\|A\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}\)

  (note this is not an induced norm)
It is said that 70% or more of applied mathematics research involves solving systems of \( m \) linear equations for \( n \) unknowns:

\[
\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1, \ldots, m.
\]

Linear systems arise directly from discrete models, e.g., traffic flow in a city. Or, they may come through representing or more abstract linear operators in some finite basis (representation). Common abstraction:

\[ Ax = b \]

Special case: Square invertible matrices, \( m = n \), \( \det A \neq 0 \):

\[ x = A^{-1}b. \]

The goal: Calculate solution \( x \) given data \( A, b \) in the most numerically stable and also efficient way.
Conditioning of linear systems

Stability analysis: rhs perturbations

Perturbations on right hand side (rhs) only:

\[ A(x + \delta x) = b + \delta b \quad \Rightarrow \quad b + A\delta x = b + \delta b \]

\[ \delta x = A^{-1}\delta b \quad \Rightarrow \quad \|\delta x\| \leq \|A^{-1}\| \|\delta b\| \]

Using the bounds

\[ \|b\| \leq \|A\| \|x\| \quad \Rightarrow \quad \|x\| \geq \|b\| / \|A\| \]

the relative error in the solution can be bounded by

\[ \frac{\|\delta x\|}{\|x\|} \leq \frac{\|A^{-1}\| \|\delta b\|}{\|x\|} \leq \frac{\|A^{-1}\| \|\delta b\|}{\|b\| / \|A\|} = \kappa(A) \frac{\|\delta b\|}{\|b\|} \]

where the **conditioning number** \( \kappa(A) \) depends on the matrix norm used:

\[ \kappa(A) = \|A\| \|A^{-1}\| \geq 1. \]
Perturbations of the matrix only:

\[(A + \delta A)(x + \delta x) = b \Rightarrow \delta x = -A^{-1}(\delta A)(x + \delta x)\]

\[\frac{\|\delta x\|}{\|x + \delta x\|} \leq \|A^{-1}\| \|\delta A\| = \kappa(A) \frac{\|\delta A\|}{\|A\|}.\]

Conclusion: The conditioning of the linear system is determined by

\[\kappa(A) = \|A\| \|A^{-1}\| \geq 1\]

No numerical method can cure an ill-conditioned systems, \(\kappa(A) \gg 1\).

The conditioning number can only be estimated in practice since \(A^{-1}\) is not available (see MATLAB's rcond function).

Practice: What is \(\kappa(A)\) for diagonal matrices in the 1-norm, \(\infty\)-norm, and 2-norm?
Now consider general perturbations of the data:

$$(A + \delta A) (x + \delta x) = b + \delta b$$

The full derivation is the book \textit{next slide}:

$$\frac{||\delta x||}{||x||} \leq \frac{\kappa(A)}{1 - \kappa(A) ||\delta A||} \left( \frac{||\delta b||}{||b||} + \frac{||\delta A||}{||A||} \right)$$

Important practical estimate:
Roundoff error in the data, with rounding unit $u$ (recall $\approx 10^{-16}$ for double precision), produces a relative error

$$\frac{||\delta x||_\infty}{||x||_\infty} \lesssim 2u\kappa(A)$$

It certainly makes no sense to try to solve systems with $\kappa(A) > 10^{16}$. 
Conditioning of linear systems

General perturbations (1)

\[(A + \delta A)(x + \delta x) = b + \delta b\]

\[b + (A + \delta A)\delta x + (\delta A)x = b + \delta b\]

\[\Rightarrow \delta x = (A + \delta A)^{-1}\left[\delta b - (\delta A)x\right]\]

\[= \left[A(I + A^{-1}\delta A)^{-1}\right]^{-1}\left[\delta b - (\delta A)x\right]\]

\[= (I + A^{-1}\delta A)^{-1}A^{-1}\left[\delta b - (\delta A)x\right]\]

\[\|\delta x\| \leq \|I + A^{-1}\delta A\|^{-1}\|\delta b - (\delta A)x\|\]

Derived in book:

Fact 1: \[\|I + A^{-1}\delta A\|^{-1} \leq \frac{1}{1 - \|A^{-1}\delta A\|} \leq \frac{1}{1 - \|A^{-1}\|\|\delta A\|}\]

Fact 2: \[\|\delta b - (\delta A)x\| \leq \|\delta b\| + \|\delta A\|\|x\| \leq \delta b + \|\delta A\|\|x\|\]
\[ \frac{\| \delta x \|}{\| x \|} \leq \frac{\| A^{-1} \|}{1 - \| A^{-1} \| \| \delta A \|} \left[ \frac{\| \delta b \|}{\| x \|} + \| \delta A \| \right] \]

\[ = \frac{\| A^{-1} \| \| A \|}{1 - \| A^{-1} \| \| A \| \| \delta A \|} \left[ \frac{\| \delta b \|}{\| A \| \| x \|} + \frac{\| \delta A \|}{\| A \|} \right] \]

\[ \leq \frac{\kappa(A) \| \delta A \|}{1 - \kappa(A) \| \delta A \| / \| A \|} \left[ \frac{\| \delta b \|}{\| \delta A \|} + \frac{\| \delta A \|}{\| A \|} \right] \]

[just \ put \ \| A \| = \text{cond}(A)]
There are several numerical methods for solving a system of linear equations.

The most appropriate method really depends on the properties of the matrix $A$:

- General **dense matrices**, where the entries in $A$ are mostly non-zero and nothing special is known. We focus on the Gaussian Elimination Method (GEM).
- General **sparse matrices**, where only a small fraction of $a_{ij} \neq 0$.
- **Symmetric** and also **positive-definite** dense or sparse matrices.
- Special **structured sparse matrices**, arising from specific physical properties of the underlying system (more in Numerical Methods II).

It is also important to consider **how many times** a linear system with the same or related matrix or right hand side needs to be solved.
GEM: Eliminating $x_1$

Step 1:

$$\begin{bmatrix}
(a_{11}) & (a_{12}) & (a_{13}) \\
(a_{21}) & (a_{22}) & (a_{23}) \\
(a_{31}) & (a_{32}) & (a_{33}) \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\end{bmatrix} =
\begin{bmatrix}
(b_1) \\
(b_2) \\
(b_3) \\
\end{bmatrix}
$$

Multiply first row by $\frac{(a_{21})}{(a_{11})}$

$$l_{21} = \frac{a_{21}}{a_{11}}$$

$$l_{31} = \frac{a_{31}}{a_{11}}$$

Eliminate $x_1$

$$\begin{bmatrix}
(a_{11}) & (a_{12}) & (a_{13}) \\
0 & a_{22} - \frac{(a_{21})}{(a_{11})} & a_{23} - \frac{(a_{21})(a_{13})}{(a_{11})} \\
0 & a_{32} - \frac{(a_{31})}{(a_{11})} & a_{33} - \frac{(a_{31})(a_{13})}{(a_{11})} \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\end{bmatrix} =
\begin{bmatrix}
(b_1) \\
(b_2 - l_{21}b_1) \\
(b_3 - l_{31}b_1) \\
\end{bmatrix}$$
Gauss elimination and LU factorization

GEM: Eliminating \( x_2 \)

Step 2:

\[
\begin{bmatrix}
a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\
a_{22}^{(2)} & a_{23}^{(2)} \\
a_{32}^{(2)} & a_{33}^{(2)}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= 
\begin{bmatrix}
b_1^{(2)} \\
b_2^{(2)} \\
b_3^{(2)}
\end{bmatrix}
\]

done row!

\[
\begin{aligned}
l_{32} &= \frac{a_{32}^{(2)}}{a_{22}^{(2)}} \\
&= \frac{a_{32}^{(2)}}{a_{22}^{(2)}}
\end{aligned}
\]

Multiply second row by \( l_{32} \)

Eliminate \( x_2 \)

\[
\begin{bmatrix}
a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\
a_{22}^{(2)} & a_{23}^{(2)} \\
a_{32}^{(3)} & a_{33}^{(3)}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= 
\begin{bmatrix}
b_1^{(3)} \\
b_2^{(3)} \\
b_3^{(3)}
\end{bmatrix}
\]

Upper triangular system

\[
\begin{aligned}
x_3 &= \frac{b_3^{(3)}}{a_{33}^{(3)}}
\end{aligned}
\]
GEM: Backward substitution

Eliminate $x_3$ entirely

$$\begin{bmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} e_1^{(3)} - a_{13} x_3 \\ e_2^{(3)} - a_{23} x_3 \end{bmatrix} = \tilde{b}$$

Solve for $x_2 = \frac{\tilde{e}_2}{a_{22}}$, then $x_1$, and done!

Idea: Store the multipliers in the lower triangle of $A$:

Matrix at Step $k$:

$$\begin{bmatrix} L^{(k)} & A^{(k)} \\ \end{bmatrix}$$
Observation, proven in the book (not very intuitively):

\[ A = LU, \]

where \( L \) is unit lower triangular \((l_{ii} = 1 \text{ on diagonal})\), and \( U \) is upper triangular.

GEM is thus essentially the same as the \textit{LU factorization method}. 
Gauss elimination and **LU** factorization

GEM in MATLAB

Sample MATLAB code (for learning purposes only, not real computing!):

```
function A = MyLU(A)
% LU factorization in-place (overwrite A)
[n,m]=size(A);
if (n ~= m); error('Matrix not square'); end
for k=1:(n-1) % For variable x(k)
    % Calculate multipliers in column k:
    A((k+1):n,k) = A((k+1):n,k) / A(k,k);
    % Note: Pivot element A(k,k) assumed nonzero!
    for j=(k+1):n
        % Eliminate variable x(k):
        A((k+1):n,j) = A((k+1):n,j) - ... 
            A((k+1):n,k) * A(k,j);
    end
end
end
```
Gauss Elimination Method (GEM)

- GEM is a **general** method for **dense matrices** and is commonly used.
- Implementing GEM efficiently is difficult and we will not discuss it here, since others have done it for you!
- The **LAPACK** public-domain library is the main repository for excellent implementations of dense linear solvers.
- MATLAB uses a highly-optimized variant of GEM by default, mostly based on LAPACK.
- MATLAB does have **specialized solvers** for special cases of matrices, so always look at the help pages!
Pivoting example

Zero diagonal entries (pivots) pose a problem → pivoting (swapping rows and columns)

\[ A \mathbf{x} = \mathbf{b} \]

\[
\begin{bmatrix}
1 & 1 & 3 \\
2 & 2 & 2 \\
3 & 6 & 4
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}
= 
\begin{bmatrix}
5 \\
6 \\
13
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 1 & 3 \\
2 & 0 & -4 \\
3 & 3 & -5
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1 & 3 \\
2 & 0 & -4 \\
3 & 3 & -5
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}
\]

Observe PERMUTED

\[ LU = A \]
GEM Matlab example (1)

>> L=[1 0 0; 3 1 0; 2 0 1]
L =
    1     0     0
    3     1     0
    2     0     1

>> U=[1 1 3; 0 3 -5; 0 0 -4]
U =
    1     1     3
    0     3    -5
    0     0    -4
Gauss elimination and LU factorization

Pivoting

GEM Matlab example (2)

\[
\begin{align*}
\text{AP} &= L \times U \quad \% \text{Permuted } A \\
\text{AP} &= \\
1 &\quad 1 &\quad 3 \\
3 &\quad 6 &\quad 4 \\
2 &\quad 2 &\quad 2 \\
\end{align*}
\]

\[
\begin{align*}
\text{A} &= \begin{bmatrix} 1 & 1 & 3 \\
2 & 2 & 2 \\
3 & 6 & 4 \end{bmatrix} \\
\text{A} &= \\
1 &\quad 1 &\quad 3 \\
2 &\quad 2 &\quad 2 \\
3 &\quad 6 &\quad 4 \\
\end{align*}
\]
GEM Matlab example (3)

```matlab
>> AP = MyLU(AP) % Two last rows permuted
AP =
    1    1     3
    3    3    -5
    2    0    -4

>> MyLU(A) % No pivoting
ans =
    1    1     3
    2    0    -4
    3    Inf    Inf
```
GEM Matlab example (4)

$$\begin{align*}
\text{Lm} &= \\
&= \begin{bmatrix}
1.0000 & 0 & 0 \\
0.6667 & 1.0000 & 0 \\
0.3333 & 0.5000 & 1.0000
\end{bmatrix} \\
\text{Um} &= \\
&= \begin{bmatrix}
3.0000 & 6.0000 & 4.0000 \\
0 & -2.0000 & -0.6667 \\
0 & 0 & 2.0000
\end{bmatrix} \\
\text{Pm} &= \\
&= \begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix}
\end{align*}$$
Gauss elimination and LU factorization

Pivoting

GEM Matlab example (5)

```matlab
>> Lm*Um
ans =
   3   6   4
   2   2   2
   1   1   3

>> A
A =
   1   1   3
   2   2   2
   3   6   4

>> norm ( Lm*Um - Pm*A )
ans =
   0
```
Partial (row) pivoting permutes the rows (equations) of $A$ in order to ensure sufficiently large pivots and thus numerical stability:

$$PA = LU$$

Here $P$ is a permutation matrix, meaning a matrix obtained by permuting rows and/or columns of the identity matrix.

Complete pivoting also permutes columns, $PAQ = LU$. 
Solving linear systems

- Once an $LU$ factorization is available, solving a linear system is simple:

$$Ax = LUx = L(Ux) = Ly = b$$

so solve for $y$ using **forward substitution**. This was implicitly done in the example above by overwriting $b$ to become $y$ during the factorization.

- Then, solve for $x$ using **backward substitution**

$$Ux = y.$$  

- In MATLAB, the **backslash operator** (see help on `mldivide`)

$$x = A\backslash b \approx A^{-1}b,$$

solves the linear system $Ax = b$ using the LAPACK library. Never use matrix inverse to do this, even if written as such on paper.
If row pivoting is necessary, the same applies if one also permutes the equations (rhs $b$):

$$PAx = LUx = Ly = Pb$$

or formally (meaning for theoretical purposes only)

$$x = (LU)^{-1} Pb = U^{-1} L^{-1} Pb$$

Observing that permutation matrices are orthogonal matrices, $P^{-1} = P^T$,

$$A = P^{-1}LU = (P^TL)U = \tilde{LU}$$

where $\tilde{L}$ is a row permutation of a unit lower triangular matrix.
Doing \( x = A \backslash b \) is **equivalent** to performing an \( LU \) factorization and doing two **triangular solves** (backward and forward substitution):

\[
[\tilde{L}, U] = lu(A) \\
y = \tilde{L} \backslash b \\
x = U \backslash y
\]

This is a carefully implemented **backward stable** pivoted \( LU \) factorization, meaning that the returned solution is as accurate as the conditioning number allows.

The MATLAB call \([L, U, P] = lu(A)\) returns the permutation matrix but the call \([\tilde{L}, U] = lu(A)\) permutes the lower triangular factor directly.
Gauss elimination and LU factorization

GEM Matlab example (1)

```matlab
>> A = [ 1  2  3 ; 4  5  6 ; 7  8  0 ];
>> b=[2  1  -1] ’;

>> x=A^(-1)*b; x ’ % Don’t do this!
ans =    −2.5556  2.1111  0.1111

>> x = A\b; x ’ % Do this instead
ans =    −2.5556  2.1111  0.1111

>> linsolve(A,b) ’ % Even more control
ans =    −2.5556  2.1111  0.1111
```
GEM Matlab example (2)

>> [L,U] = lu(A) % Even better if resolving

L =
0.1429  1.0000  0
 0.5714  0.5000  1.0000
 1.0000  0   0

U =
7.0000  8.0000  0
 0   0.8571  3.0000
 0   0   4.5000

>> norm(L*U-A, inf)
ans =
0

>> y = L\b;
>> x = U\y; x'
ans =
-2.5556  2.1111  0.1111
Cost estimates for GEM

- For forward or backward substitution, at step $k$ there are $\sim (n - k)$ multiplications and subtractions, plus a few divisions. The total over all $n$ steps is

$$\sum_{k=1}^{n} (n - k) = \frac{n(n - 1)}{2} \approx \frac{n^2}{2}$$

subtractions and multiplications, giving a total of $n^2$ floating-point operations (FLOPs).

- For GEM, at step $k$ there are $\sim (n - k)^2$ multiplications and subtractions, plus a few divisions. The total is

$$\text{FLOPS} = 2 \sum_{k=1}^{n} (n - k)^2 \approx \frac{2n^3}{3},$$

and the $O(n^2)$ operations for the triangular solves are neglected.

- When many linear systems need to be solved with the same $A$ the factorization can be reused.
A real symmetric matrix $A$ is positive definite iff (if and only if):

1. All of its eigenvalues are real (follows from symmetry) and positive.
2. $\forall x \neq 0, x^T A x > 0$, i.e., the quadratic form defined by the matrix $A$ is convex.
3. There exists a unique lower triangular $L$, $L_{ii} > 0$,

$$A = LL^T,$$

termed the Cholesky factorization of $A$ (symmetric $LU$ factorization).

For Hermitian complex matrices just replace transposes with adjoints (conjugate transpose), e.g., $A^T \rightarrow A^*$ (or $A^H$ in the book).
The MATLAB built-in function

\[ R = \text{chol}(A) \]

gives the Cholesky factorization and is a good way to test for positive-definiteness.

For Hermitian/symmetric matrices with positive diagonals MATLAB tries a Cholesky factorization first, before resorting to LU factorization with pivoting.

The cost of a Cholesky factorization is about half the cost of GEM, \( n^3/3 \) FLOPS.
When pivoting is unnecessary

- It can be shown that roundoff is **not** a problem for triangular system $Tx = b$ (forward or backward substitution). Specifically,

$$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \lesssim nuk(T),$$

so unless the number of unknowns $n$ is very very large the truncation errors are small for **well-conditioned systems**.

- Special classes of **well-behaved** matrices $A$:
  1. **Diagonally-dominant** matrices, meaning
     $$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \text{ or } |a_{ii}| \geq \sum_{j \neq i} |a_{ji}|$$

  2. **Symmetric positive-definite** matrices, i.e., Cholesky factorization does not require pivoting,
     $$\frac{\|\delta x\|_2}{\|x\|_2} \lesssim 8n^2 u\kappa(A).$$
When pivoting is necessary

- For a general matrix $A$, roundoff analysis leads to the following type of estimate
  $$\frac{\|\delta x\|}{\|x\|} \lesssim n u \kappa(A) \frac{\|L\| \|U\|}{\|A\|},$$
  which shows that small pivots, i.e., large multipliers $l_{ij}$, can lead to large roundoff errors.

  What we want is an estimate that **only** involves $n$ and $\kappa(A)$.

- Since the optimal pivoting **cannot** be predicted a-priori, it is best to **search for the largest pivot in the same column as the current pivot**, and exchange the two rows (partial pivoting).
Partial Pivoting

- The cost of partial pivoting is searching among $O(n)$ elements $n$ times, so $O(n^2)$, which is small compared to $O(n^3)$ total cost.
- Complete pivoting requires searching $O(n^2)$ elements $n$ times, so cost is $O(n^3)$ which is usually not justified.
- The recommended strategy is to use partial (row) pivoting even if not strictly necessary (MATLAB takes care of this).
What pivoting does

- The problem with GEM without pivoting is large growth factors (not large numbers per se)

\[ \rho = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} \]

- Pivoting is not needed for positive-definite matrices because \( \rho \leq 2 \):

\[ |a_{ij}|^2 \leq |a_{ii}| |a_{jj}| \] (so the largest element is on the diagonal)

\[ a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)} = a_{ij}^{(k)} - \frac{a_{ki}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} \] (GEM)

\[ a_{ii}^{(k+1)} = a_{ii}^{(k)} - \left( \frac{a_{ki}^{(k)}}{a_{kk}^{(k)}} \right)^2 \quad \Rightarrow \quad |a_{ii}^{(k+1)}| \leq |a_{ii}^{(k)}| + \left| \frac{a_{ki}^{(k)}}{a_{kk}^{(k)}} \right|^2 \leq 2 |a_{ii}^{(k)}| \]
Pivoting is not always sufficient to ensure lack of roundoff problems. In particular, \textbf{large variations} among the entries in $A$ should be avoided.

This can usually be remedied by changing the physical units for $x$ and $b$ to be the \textbf{natural units} $x_0$ and $b_0$.

\textbf{Rescaling} the unknowns and the equations is generally a good idea even if not necessary:

$$x = D_x \tilde{x} = \text{Diag} \{x_0\} \tilde{x} \quad \text{and} \quad b = D_b \tilde{b} = \text{Diag} \{b_0\} \tilde{b}.$$  

$$Ax = AD_x \tilde{x} = D_b \tilde{b} \quad \Rightarrow \quad (D_b^{-1} AD_x) \tilde{x} = \tilde{b}$$

The \textbf{rescaled matrix} $\tilde{A} = D_b^{-1} AD_x$ should have a better conditioning, but this is hard to achieve in general.

Also note that \textbf{reordering the variables} from most important to least important may also help.
MATLAB recognizes (i.e., tests for) some special matrices automatically: banded, permuted lower/upper triangular, symmetric, Hessenberg, but not sparse.

In MATLAB one may specify a matrix $B$ instead of a single right-hand side vector $b$.

The MATLAB function

$$X = \text{linsolve}(A, B, \text{opts})$$

allows one to specify certain properties that speed up the solution (triangular, upper Hessenberg, symmetric, positive definite, none), and also estimates the condition number along the way.

Use $\text{linsolve}$ instead of backslash if you know (for sure!) something about your matrix.
The conditioning of a linear system $Ax = b$ is determined by the condition number

$$\kappa(A) = \|A\|\|A^{-1}\| \geq 1$$

Gauss elimination can be used to solve general square linear systems and also produces a factorization $A = LU$.

Partial pivoting is often necessary to ensure numerical stability during GEM and leads to $PA = LU$ or $A = \tilde{LU}$.

For symmetric positive definite matrices the Cholesky factorization $A = LL^T$ is preferred and does not require pivoting.

MATLAB has excellent linear solvers based on well-known public domain libraries like LAPACK. Use them!