Numerical Methods I
Solving Square Linear Systems: GEM and LU factorization

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

[1]MATH-GA 2011.003 / CSCI-GA 2945.003, Fall 2014

September 18th, 2014

Outline

1. Linear Algebra Background
2. Conditioning of linear systems
3. Gauss elimination and **LU** factorization
   - Pivoting
   - LU factorization
   - Cholesky Factorization
   - Pivoting and Stability
4. Conclusions

Kernel Space

- The dimension of the column space of a matrix is called the **rank** of the matrix $\mathbf{A} \in \mathbb{R}^{m,n}$,

$$r = \operatorname{rank} \mathbf{A} \leq \min(m, n).$$

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

$$\mathbf{Ax} = \mathbf{0}.$$

- 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 $\mathbf{A} \in \mathbb{R}^{m,n}$

$$\operatorname{rank} \mathbf{A} + \operatorname{nullity} \mathbf{A} = n.$$

- 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, $\operatorname{im} \mathbf{A}^T$.
    Its dimension is also equal to the the rank.
  - **Left nullspace** or **cokernel** of a matrix is the nullspace or kernel of its transpose, $\ker \mathbf{A}^T$.

- Second fundamental theorem in linear algebra:

$$\operatorname{im} \mathbf{A}^T = (\ker \mathbf{A})^\perp$$

$$\ker \mathbf{A}^T = (\operatorname{im} \mathbf{A})^\perp$$

The Matrix Inverse

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

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I},$$

  where $\mathbf{I}$ is the identity matrix (ones along diagonal, all the rest zeros).
- The following statements are equivalent for $\mathbf{A} \in \mathbb{R}^{n,n}$:
  - $\mathbf{A}$ is **invertible**.
  - $\mathbf{A}$ is **full-rank**, $\operatorname{rank} \mathbf{A} = n$.
  - The columns and also the rows are linearly independent and form a **basis** for $\mathbb{R}^n$.
  - The **determinant** is nonzero, $\det \mathbf{A} \neq 0$.
  - Zero is not an eigenvalue of $\mathbf{A}$.

Matrix Algebra

- Matrix-matrix multiplication is **not commutative**, $\mathbf{AB} \neq \mathbf{BA}$ in general. Note $\mathbf{x}^T\mathbf{y}$ is a scalar (dot product) so this commutes.
- Some useful properties:

$$\mathbf{C}(\mathbf{A}+\mathbf{B}) = \mathbf{CA}+\mathbf{CB} \text{ and } \mathbf{ABC} = (\mathbf{AB})\,\mathbf{C} = \mathbf{A}(\mathbf{BC})$$

$$\left(\mathbf{A}^T\right)^T = \mathbf{A} \text{ and } (\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$$

$$\left(\mathbf{A}^{-1}\right)^{-1} = \mathbf{A} \text{ and } (\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1} \text{ and } \left(\mathbf{A}^T\right)^{-1} = \left(\mathbf{A}^{-1}\right)^T$$

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

$$\mathbf{AB} = \mathbf{C} \quad \Rightarrow \quad \left(\mathbf{A}^{-1}\mathbf{A}\right)\mathbf{B} = \mathbf{A}^{-1}\mathbf{C} \quad \Rightarrow \quad \begin{cases} \mathbf{B} & = \mathbf{A}^{-1}\mathbf{C} \\ \mathbf{A} & = \mathbf{CB}^{-1} \end{cases}$$

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:

1. The 1-norm (L^1 norm or Manhattan distance), $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$
2. The 2-norm (L^2 norm, **Euclidian distance**), $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x} \cdot \mathbf{x}} = \sqrt{\sum_{i=1}^{n} |x_i|^2}$
3. The ∞-norm (L^∞ or maximum norm), $\|\mathbf{x}\|_\infty = \max_{1 \le i \le n} |x_i|$

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:

$$\|\mathbf{A}\| = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{Ax}\|}{\|\mathbf{x}\|} \quad \Rightarrow \|\mathbf{Ax}\| \leq \|\mathbf{A}\| \, \|\mathbf{x}\|$$

- The last bound holds for matrices as well, $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \, \|\mathbf{B}\|$.
- Special cases of interest are:
  1. The 1-norm or **column sum norm**, $\|\mathbf{A}\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$
  2. The ∞-norm or **row sum norm**, $\|\mathbf{A}\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|$
  3. The 2-norm or **spectral norm**, $\|\mathbf{A}\|_2 = \sigma_1$ (largest singular value)
  4. The Euclidian or **Frobenius norm**, $\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$ (note this is not an induced norm)

Matrices and linear systems

- It is said that 70% or more of applied mathematics research involves solving systems of m linear equations for n unknowns:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1, \cdots, m.$$

- 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:

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

- Special case: Square invertible matrices, $m = n$, $\det \mathbf{A} \neq 0$:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}.$$

- The goal: Calculate solution $\mathbf{x}$ given data $\mathbf{A}, \mathbf{b}$ in the most numerically stable and also efficient way.

Stability analysis: rhs perturbations

Perturbations on right hand side (rhs) only:

$$\mathbf{A}\left(\mathbf{x}+\delta\mathbf{x}\right)=\mathbf{b}+\delta\mathbf{b}\quad\Rightarrow\mathbf{b}+\mathbf{A}\delta\mathbf{x}=\mathbf{b}+\delta\mathbf{b}$$

$$\delta\mathbf{x}=\mathbf{A}^{-1}\delta\mathbf{b}\quad\Rightarrow\left\|\delta\mathbf{x}\right\|\leq\left\|\mathbf{A}^{-1}\right\|\left\|\delta\mathbf{b}\right\|$$

Using the bounds

$$\left\|\mathbf{b}\right\|\leq\left\|\mathbf{A}\right\|\left\|\mathbf{x}\right\|\quad\Rightarrow\left\|\mathbf{x}\right\|\geq\left\|\mathbf{b}\right\|/\left\|\mathbf{A}\right\|$$

the relative error in the solution can be bounded by

$$\frac{\left\|\delta\mathbf{x}\right\|}{\left\|\mathbf{x}\right\|}\leq\frac{\left\|\mathbf{A}^{-1}\right\|\left\|\delta\mathbf{b}\right\|}{\left\|\mathbf{x}\right\|}\leq\frac{\left\|\mathbf{A}^{-1}\right\|\left\|\delta\mathbf{b}\right\|}{\left\|\mathbf{b}\right\|/\left\|\mathbf{A}\right\|}=\kappa(\mathbf{A})\frac{\left\|\delta\mathbf{b}\right\|}{\left\|\mathbf{b}\right\|}$$

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

$$\kappa(\mathbf{A})=\left\|\mathbf{A}\right\|\left\|\mathbf{A}^{-1}\right\|\geq 1.$$

Stability analysis: matrix perturbations

- Perturbations of the matrix only:

$$(\mathbf{A} + \delta\mathbf{A})(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} \quad \Rightarrow \delta\mathbf{x} = -\mathbf{A}^{-1}(\delta\mathbf{A})(\mathbf{x} + \delta\mathbf{x})$$

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x} + \delta\mathbf{x}\|} \leq \left\|\mathbf{A}^{-1}\right\| \|\delta\mathbf{A}\| = \kappa(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}.$$

- Conclusion: The conditioning of the linear system is determined by

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

- No numerical method can cure an ill-conditioned systems, $\kappa(\mathbf{A}) \gg 1$.
- The conditioning number can only be **estimated** in practice since $\mathbf{A}^{-1}$ is not available (see MATLAB's *rcond* function).

Practice: What is $\kappa(\mathbf{A})$ for diagonal matrices in the 1-norm, ∞-norm, and 2-norm?

Mixed perturbations

- Now consider general perturbations of the data:

$$(\mathbf{A}+\delta\mathbf{A})(\mathbf{x}+\delta\mathbf{x})=\mathbf{b}+\delta\mathbf{b}$$

- The full derivation is the book [*next slide*]:

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

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

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

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

General perturbations (1)

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

$$\cancel{b} + (A+\delta A)\delta x + (\delta A)x = \cancel{b} + \delta b$$

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

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

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

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

Derived in book:

FACT 1: $\|(I + A^{-1}\delta A)^{-1}\| \leq \dfrac{1}{1 - \|A^{-1}\delta A\|} \leq \dfrac{1}{1 - \|A^{-1}\|\|\delta A\|}$

FACT 2: $\|\delta b - (\delta A)x\| \leq \|\delta b\| + \|(\delta A)x\| \leq \delta b + \|\delta A\|\|\delta x\|$ (1)

General perturbations (2)

$$\Rightarrow \quad \frac{\|\delta x\|}{\|x\|} \le \frac{\|A^{-1}\|}{1 - \|A^{-1}\| \|\delta A\|} \cdot \left[\frac{\|\delta b\|}{\|x\|} + \|\delta A\| \right]$$

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

[just put $\|A\|$ in both numerator and denom.]

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

2

Numerical Solution of Linear Systems

- 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: $\quad Ax = b$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} \begin{array}{l} \leftarrow \text{Multiply FIRST row by} \\ \leftarrow l_{21} = \dfrac{a_{21}^{(1)}}{a_{11}^{(1)}} \\ \leftarrow l_{31} = \dfrac{a_{31}^{(1)}}{a_{11}^{(1)}} \end{array}$$

$\Downarrow$ Eliminate x_1

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 = a_{21}^{(1)} - l_{21} \cdot a_{11} & a_{22}^{(1)} - l_{21} \cdot a_{12}^{(1)} & a_{23}^{(1)} - l_{21} \cdot a_{13}^{(1)} \\ 0 & a_{32}^{(1)} - l_{31} \cdot a_{12}^{(1)} & a_{33}^{(1)} - l_{31} \cdot a_{13}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 - l_{21} \cdot b_1 \\ b_3 - l_{31} \cdot b_1 \end{bmatrix}$$

1

GEM: Eliminating x_2

Step 2:

$$
\left[\begin{array}{c|cc}
a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\
\hline
0 & a_{22}^{(2)} & a_{23}^{(2)} \\
0 & a_{32}^{(2)} & a_{33}^{(2)}
\end{array}\right]
\left[\begin{array}{c}
x_1 \\ \hline x_2 \\ \hline x_3
\end{array}\right]
=
\left[\begin{array}{c}
b_1^{(2)} \\ \hline b_2^{(2)} \\ \hline b_3^{(2)}
\end{array}\right]
\begin{array}{l}
\text{done row!} \\
\leftarrow \text{Multiply second row by} \\
\leftarrow l_{32} = \dfrac{a_{32}^{(2)}}{a_{22}^{(2)}}
\end{array}
$$

$\Downarrow$ Eliminate x_2

$$
\left[\begin{array}{c|c|c}
a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\
\hline
0 & a_{22}^{(2)} & a_{23}^{(2)} \\
\hline
0 & 0 & a_{33}^{(3)}
\end{array}\right]
\left[\begin{array}{c}
x_1 \\ x_2 \\ x_3
\end{array}\right]
=
\left[\begin{array}{c}
b_1^{(3)} \\ \hline b_2^{(3)} \\ \hline b_3^{(3)}
\end{array}\right]
\begin{array}{l}
\text{Upper triangular system} \\
\\
\leftarrow \text{Solve } x_3 = \dfrac{b_3^{(3)}}{a_{33}^{(3)}}
\end{array}
$$

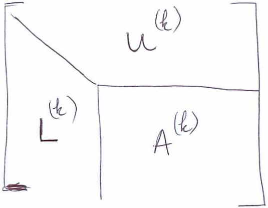
GEM: Backward substitution

Eliminate x_3 entirely $\rightarrow$
$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} \\ 0 & a_{22}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1^{(3)} - a_{13}^{(1)} x_3 \\ b_2^{(3)} - a_{23}^{(2)} x_3 \end{bmatrix} = \tilde{b}$$

solve for $x_2 = \dfrac{\tilde{b}}{a_{22}^{(2)}}$, then x_1, and done!

IDEA: Store the multipliers in the lower triangle of A:

Matrix at Step k:

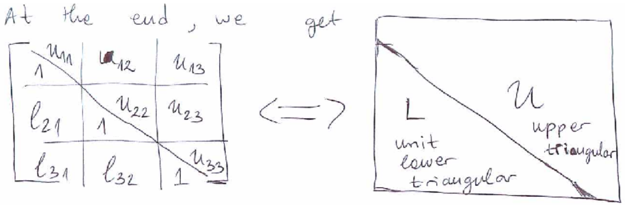


$$\left[\begin{array}{c|c|c} u_{11} & u_{12} & u_{13} \\ \hline l_{21} & a_{22}^{(2)} & a_{23}^{(2)} \\ \hline l_{31} & a_{32}^{(2)} & a_{33}^{(2)} \end{array}\right]$$

Example step 2

3

GEM as an LU factorization tool



- Observation, proven in the book (not very intuitively):

$$\mathbf{A} = \mathbf{LU},$$

  where **L** is **unit lower triangular** ($l_{ii} = 1$ on diagonal), and **U** is **upper triangular**.
- GEM is thus essentially the same as the LU **factorization method**.

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 $\longrightarrow$ PIVOTING (swapping rows and columns)

$$Ax = b$$

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 13 \end{bmatrix} \Rightarrow \left[\begin{array}{c|cc} 1 & 1 & 3 \\ \hline 2 & 0 & -4 \\ 3 & 3 & -5 \end{array}\right]$$

$$\left[\begin{array}{c|cc} 1 & 1 & 3 \\ \hline 3 & 3 & -5 \\ 2 & 0 & -4 \end{array}\right] \Rightarrow \left[\begin{array}{c|c|c} {}_{1}\;1 & 1 & 3 \\ \hline 3 & {}_{1}\;3 & -5 \\ \hline 2 & 0 & {}_{1}\;-4 \end{array}\right]$$

OBSERVE

$$LU = \text{PERMUTED } A$$

4

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
```

GEM Matlab example (2)

```
>> AP=L*U % Permuted A
AP =

     1     1     3
     3     6     4
     2     2     2

>> A=[1 1 3; 2 2 2; 3 6 4]
A =
     1     1     3
     2     2     2
     3     6     4
```

GEM Matlab example (3)

```
>> 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
```

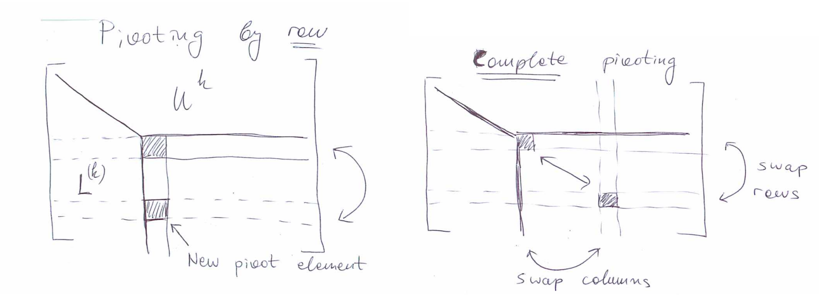
GEM Matlab example (4)

```
>> [Lm,Um,Pm]=lu(A)
Lm =
    1.0000         0         0
    0.6667    1.0000         0
    0.3333    0.5000    1.0000
Um =
    3.0000    6.0000    4.0000
         0   -2.0000   -0.6667
         0         0    2.0000
Pm =
     0     0     1
     0     1     0
     1     0     0
```

GEM Matlab example (5)

```
>> 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
```

Pivoting during **LU** factorization



- **Partial (row) pivoting** permutes the rows (equations) of **A** in order to ensure sufficiently large pivots and thus numerical stability:

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

- 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, $\mathbf{PAQ} = \mathbf{LU}$.

Solving linear systems

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

$$\mathbf{Ax} = \mathbf{LUx} = \mathbf{L}(\mathbf{Ux}) = \mathbf{Ly} = \mathbf{b}$$

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

- Then, solve for $\mathbf{x}$ using **backward substitution**

$$\mathbf{Ux} = \mathbf{y}.$$

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

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

  solves the linear system $\mathbf{Ax} = \mathbf{b}$ using the LAPACK library.
  Never use matrix inverse to do this, even if written as such on paper.

Permutation matrices

- If row pivoting is necessary, the same applies if one also permutes the equations (rhs **b**):

$$\mathbf{PAx} = \mathbf{LUx} = \mathbf{Ly} = \mathbf{Pb}$$

  or *formally* (meaning for theoretical purposes only)

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

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

$$\mathbf{A} = \mathbf{P}^{-1}\mathbf{LU} = \left(\mathbf{P}^{T}\mathbf{L}\right)\mathbf{U} = \widetilde{\mathbf{L}}\mathbf{U}$$

  where $\widetilde{\mathbf{L}}$ is a row permutation of a unit lower triangular matrix.

In MATLAB

- Doing $x = A \backslash b$ is **equivalent** to performing an LU factorization and doing two **triangular solves** (backward and forward substitution):

$$
\begin{aligned}
[\tilde{L}, U] &= lu(A) \\
y &= \tilde{L} \backslash b \\
x &= U \backslash y
\end{aligned}
$$

- 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.

GEM Matlab example (1)

```
>> 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 $\mathbf{A}$ the **factorization can be reused**.

Positive-Definite Matrices

- A real symmetric matrix $\mathbf{A}$ is positive definite iff (if and only if):
  1. All of its eigenvalues are real (follows from symmetry) and positive.
  2. $\forall x \neq \mathbf{0}$, $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$, i.e., the quadratic form defined by the matrix $\mathbf{A}$ is convex.
  3. There exists a *unique* lower triangular $\mathbf{L}$, $L_{ii} > 0$,
     $$\mathbf{A} = \mathbf{L}\mathbf{L}^T,$$
     termed the **Cholesky factorization** of $\mathbf{A}$ (symmetric LU factorization).

1. For Hermitian complex matrices just replace transposes with adjoints (conjugate transpose), e.g., $\mathbf{A}^T \to \mathbf{A}^\star$ (or $\mathbf{A}^H$ in the book).

Cholesky Factorization

- The MATLAB built in function

$$R = chol(A)$$

  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 $\mathbf{Tx} = \mathbf{b}$ (forward or backward substitution). Specifically,

$$\frac{\|\delta\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} \lesssim nu\kappa(\mathbf{T}),$$

  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 $\mathbf{A}$:
  1. **Diagonally-dominant** matrices, meaning

$$|a_{ii}| \geq \sum_{j\neq i} |a_{ij}| \quad \text{or} \quad |a_{ii}| \geq \sum_{j\neq i} |a_{ji}|$$

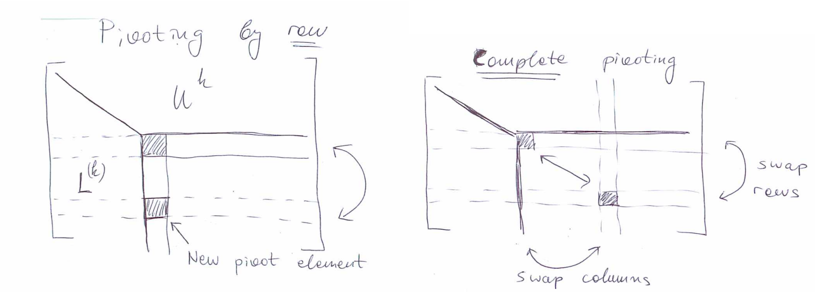
  2. **Symmetric positive-definite** matrices, i.e., Cholesky factorization does not require pivoting,

$$\frac{\|\delta\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \lesssim 8n^2 u\kappa(\mathbf{A}).$$

When pivoting is necessary

- For a general matrix $\mathbf{A}$, roundoff analysis leads to the following type of estimate
$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \lesssim nu\kappa(\mathbf{A})\frac{\||\mathbf{L}|\,|\mathbf{U}|\|}{\|\mathbf{A}\|},$$
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 $\kappa(\mathbf{A})$.
- 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)

$$\rho = \frac{\max_{i,j,k} \left| a_{ij}^{(k)} \right|}{\max_{i,j} \left| a_{ij} \right|}$$

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

$$\left| a_{ij} \right|^2 \leq \left| a_{ii} \right| \left| a_{jj} \right| \text{ (so the largest element is on the diagonal)}$$

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

$$a_{ii}^{(k+1)} = a_{ii}^{(k)} - \frac{\left(a_{ki}^{(k)} \right)^2}{a_{kk}^{(k)}} \quad \Rightarrow \left| a_{ii}^{(k+1)} \right| \leq \left| a_{ii}^{(k)} \right| + \frac{\left| a_{ki}^{(k)} \right|^2}{\left| a_{kk}^{(k)} \right|} \leq 2 \left| a_{ii}^{(k)} \right|$$

Matrix Rescaling

- Pivoting is not always sufficient to ensure lack of roundoff problems. In particular, **large variations** among the entries in **A should be avoided**.
- This can usually be remedied by changing the physical units for $\mathbf{x}$ and $\mathbf{b}$ to be the **natural units** $\mathbf{x}_0$ and $\mathbf{b}_0$.
- **Rescaling** the unknowns and the equations is generally a good idea even if not necessary:

$$\mathbf{x} = \mathbf{D}_x \tilde{\mathbf{x}} = \text{Diag}\left\{\mathbf{x}_0\right\} \tilde{\mathbf{x}} \text{ and } \mathbf{b} = \mathbf{D}_b \tilde{\mathbf{b}} = \text{Diag}\left\{\mathbf{b}_0\right\} \tilde{\mathbf{b}}.$$

$$\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{D}_x \tilde{\mathbf{x}} = \mathbf{D}_b \tilde{\mathbf{b}} \quad \Rightarrow \quad \left(\mathbf{D}_b^{-1}\mathbf{A}\mathbf{D}_x\right) \tilde{\mathbf{x}} = \tilde{\mathbf{b}}$$

- The **rescaled matrix** $\widetilde{\mathbf{A}} = \mathbf{D}_b^{-1}\mathbf{A}\mathbf{D}_x$ should have a better conditioning, but this is hard to achieve in general.
- Also note that **reordering the variables** from most important to least important may also help.

Special Matrices in MATLAB

- 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

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

  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 *linsolve* instead of backslash if you know (for sure!) something about your matrix.

Conclusions/Summary

- The conditioning of a linear system $\mathbf{Ax} = \mathbf{b}$ is determined by the condition number
$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \left\|\mathbf{A}^{-1}\right\| \geq 1$$
- Gauss elimination can be used to solve general square linear systems and also produces a factorization $\mathbf{A} = \mathbf{LU}$.
- Partial pivoting is often necessary to ensure numerical stability during GEM and leads to $\mathbf{PA} = \mathbf{LU}$ or $\mathbf{A} = \widetilde{\mathbf{L}}\mathbf{U}$.
- For symmetric positive definite matrices the Cholesky factorization $\mathbf{A} = \mathbf{LL}^T$ is preferred and does not require pivoting.
- MATLAB has excellent linear solvers based on well-known public domain libraries like LAPACK. Use them!