Scientific Computing:  
Dense Linear Systems

Aleksandar Donev  
Courant Institute, NYU\textsuperscript{1}  
donev@courant.nyu.edu

\textsuperscript{1}\textsuperscript{Course MATH-GA.2043 or CSCI-GA.2112, Spring 2012

February 9th, 2012
Outline

1. Linear Algebra Background
2. Conditioning of linear systems
3. Gauss elimination and LU factorization
4. Conclusions
A **vector space** \( \mathcal{V} \) is a set of elements called **vectors** \( \mathbf{x} \in \mathcal{V} \) that may be multiplied by a **scalar** \( c \) and added, e.g.,

\[
\mathbf{z} = \alpha \mathbf{x} + \beta \mathbf{y}
\]

I will denote scalars with lowercase letters and vectors with lowercase bold letters.

Prominent examples of vector spaces are \( \mathbb{R}^n \) (or more generally \( \mathbb{C}^n \)), but there are many others, for example, the set of polynomials in \( x \).

A **subspace** \( \mathcal{V}' \subseteq \mathcal{V} \) of a vector space is a subset such that sums and multiples of elements of \( \mathcal{V}' \) remain in \( \mathcal{V}' \) (i.e., it is closed).

An example is the set of vectors in \( x \in \mathbb{R}^3 \) such that \( x_3 = 0 \).
Consider a set of $n$ vectors $a_1, a_2, \cdots, a_n \in \mathbb{R}^m$ and form a matrix by putting these vectors as columns

$$A = [a_1 | a_2 | \cdots | a_m] \in \mathbb{R}^{m,n}.$$ 

I will denote matrices with bold capital letters, and sometimes write $A = [m, n]$ to indicate dimensions.

The matrix-vector product is defined as a linear combination of the columns:

$$b = Ax = x_1a_1 + x_2a_2 + \cdots + x_na_n \in \mathbb{R}^m.$$ 

The image $\text{im}(A)$ or range $\text{range}(A)$ of a matrix is the subspace of all linear combinations of its columns, i.e., the set of all $b$'s. It is also sometimes called the column space of the matrix.
The set of vectors $a_1, a_2, \ldots, a_n$ are \textbf{linearly independent} or form a \textbf{basis} for $\mathbb{R}^m$ if $b = Ax = 0$ implies that $x = 0$.

The \textbf{dimension} $r = \dim \mathcal{V}$ of a vector (sub)space $\mathcal{V}$ is the number of elements in a basis. This is a property of $\mathcal{V}$ itself and \textit{not} of the basis, for example,

$$\dim \mathbb{R}^n = n$$

Given a basis $\mathbf{A}$ for a vector space $\mathcal{V}$ of dimension $n$, every vector of $b \in \mathcal{V}$ can be uniquely represented as the vector of coefficients $x$ in that particular basis,

$$b = x_1 a_1 + x_2 a_2 + \cdots + x_n a_n.$$ 

A simple and common basis for $\mathbb{R}^n$ is $\{e_1, \ldots, e_n\}$, where $e_k$ has all components zero except for a single 1 in position $k$. With this choice of basis the coefficients are simply the entries in the vector, $b \equiv x$. 
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.
- For a basis $A$ the nullspace is $\text{null}(A) = \{0\}$ and the nullity is zero.
Orthogonal Spaces

- An inner-product space is a vector space together with an inner or dot product, which must satisfy some properties.
- The standard dot-product in $\mathbb{R}^n$ is denoted with several different notations:
  \[ x \cdot y = (x, y) = \langle x, y \rangle = x^T y = \sum_{i=1}^{n} x_i y_i. \]
- For $\mathbb{C}^n$ we need to add complex conjugates (here $\ast$ denotes a complex conjugate transpose, or adjoint),
  \[ x \cdot y = x^\ast y = \sum_{i=1}^{n} \bar{x}_i y_i. \]
- Two vectors $x$ and $y$ are orthogonal if $x \cdot y = 0$. 
One of the most important theorems in linear algebra is that the sum of rank and nullity is equal to the number of columns: For $A \in \mathbb{R}^{m,n}$

$$\text{rank } A + \text{nullity } A = n.$$ 

In addition to the range and kernel spaces of a matrix, two more important vector subspaces for a given matrix $A$ are the:

- **Row space** or **coimage** of a matrix is the column (image) space of its transpose, $\text{im } A^T$.  
  *Its dimension is also equal to the the rank.* 

- **Left nullspace** or **cokernel** of a matrix is the nullspace or kernel of its transpose, $\text{ker } A^T$. 

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 \( \mathcal{V} \).

Let \( \mathcal{V} \) be the set of vectors in \( x \in \mathbb{R}^3 \) such that \( x_3 = 0 \). Then \( \mathcal{V}^\perp \) is the set of all vectors with \( x_1 = x_2 = 0 \).

Second fundamental theorem in linear algebra:

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

\[
\ker A^T = (\text{im } A)^\perp
\]
A function $L : \mathcal{V} \rightarrow \mathcal{W}$ mapping from a vector space $\mathcal{V}$ to a vector space $\mathcal{W}$ is a **linear function** or a **linear transformation** if

$$L(\alpha \mathbf{v}) = \alpha L(\mathbf{v}) \text{ and } L(\mathbf{v}_1 + \mathbf{v}_2) = L(\mathbf{v}_1) + L(\mathbf{v}_2).$$

Any linear transformation $L$ can be represented as a multiplication by a matrix $L$

$$L(\mathbf{v}) = L\mathbf{v}.$$

For the common bases of $\mathcal{V} = \mathbb{R}^n$ and $\mathcal{W} = \mathbb{R}^m$, the product $\mathbf{w} = L\mathbf{v}$ is simply the usual **matrix-vector product**, 

$$w_i = \sum_{k=1}^{n} L_{ik} v_k,$$

which is simply the dot-product between the $i$-th row of the matrix and the vector $\mathbf{v}$. 
Matrix algebra

\( w_i = (Lv)_i = \sum_{k=1}^{n} L_{ik} v_k \)

- The composition of two linear transformations \( A = [m, p] \) and \( B = [p, n] \) is a **matrix-matrix product** \( C = AB = [m, n] \):

\[
z = A (Bx) = Ay = (AB)x
\]

\[
z_i = \sum_{k=1}^{n} A_{ik} y_k = \sum_{k=1}^{p} A_{ik} \sum_{j=1}^{n} B_{kj} x_j = \sum_{j=1}^{n} \left( \sum_{k=1}^{p} A_{ik} B_{kj} \right) x_j = \sum_{j=1}^{n} C_{ij} x_j
\]

\[
C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}
\]

- Matrix-matrix multiplication is **not commutative**, \( AB \neq BA \) in general.
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 $A$. 
Matrix Algebra

- Matrix-vector multiplication is just a special case of matrix-matrix multiplication. Note $\mathbf{x}^T \mathbf{y}$ is a scalar (dot product).

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

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

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

- 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 $x \in \mathbb{R}^n$, the $p$-norm is

$$
\|x\|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p}
$$

and special cases of interest are:

1. The 1-norm ($L^1$ norm or Manhattan distance), $\|x\|_1 = \sum_{i=1}^{n} |x_i|$
2. The 2-norm ($L^2$ norm, Euclidian distance),

$$
\|x\|_2 = \sqrt{x \cdot x} = \sqrt{\sum_{i=1}^{n} |x_i|^2}
$$
3. The $\infty$-norm ($L^\infty$ or maximum norm), $\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$

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

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

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

- 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)
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:

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

- 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 \textbf{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 worst-case conditioning of the linear system is determined by

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

Solving an ill-conditioned system, \(\kappa(A) \gg 1\), should only be done if something is known about \(x\) or \(b\) (e.g., smooth solutions in PDEs).

The conditioning number can only be estimated in practice since \(A^{-1}\) is not available (see MATLAB’s \(rcond\) function).
Now consider general perturbations of the data:

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

The full derivation, not given here, estimates the uncertainty or perturbation in the solution:

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

Best possible error with rounding unit $u u \approx 10^{-16}$:

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

which is close to what the MATLAB linear solvers achieve.

It certainly makes no sense to try to solve systems with $\kappa(A) > 10^{15}$!
GEM: Eliminating $x_1$

Step 1:

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

- Multiply first row by $\frac{b_1^{(1)}}{a_{11}^{(1)}}$
- $l_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}}$
- $l_{31} = \frac{a_{31}^{(1)}}{a_{11}^{(1)}}$

Eliminate $x_1$:

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

Step 2:

\[
\begin{bmatrix}
  a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\
  0 & a_{22}^{(2)} & a_{23}^{(2)} \\
  0 & a_{32}^{(2)} & a_{33}^{(2)} \\
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
\end{bmatrix}
= \begin{bmatrix}
  b_1^{(2)} \\
  b_2^{(2)} \\
  b_3^{(5)} \\
\end{bmatrix}
\]

Eliminate $x_2$

\[
\begin{bmatrix}
  a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\
  0 & a_{22}^{(2)} & a_{23}^{(2)} \\
  0 & 0 & a_{33}^{(3)} \\
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
\end{bmatrix}
= \begin{bmatrix}
  b_1^{(3)} \\
  b_2^{(3)} \\
  b_3^{(3)} \\
\end{bmatrix}
\]

Upper triangular system

\[
x_3 = \frac{b_3^{(3)}}{a_{33}^{(3)}}
\]

\[
x_2 = \frac{b_2^{(3)} - a_{23}^{(2)}x_3}{a_{22}^{(2)}}
\]

\[
x_1 = \frac{b_1^{(3)} - a_{12}^{(1)}x_2 - a_{13}^{(1)}x_3}{a_{11}^{(1)}}
\]
GEM: Backward substitution

\[
\begin{bmatrix}
  a_{11}^{(1)} & a_{12}^{(1)} \\
  0 & a_{22}^{(2)}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2
\end{bmatrix} =
\begin{bmatrix}
  e_1^{(3)} - a_{13}^{(1)} x_3 \\
  e_2^{(3)} - a_{23}^{(2)} x_3
\end{bmatrix} = \tilde{\mathbf{b}}
\]

Solve for \( x_2 = \frac{\tilde{e}_2}{a_{22}^{(2)}} \), then \( x_1 \), and done!

Idea: Store the multipliers in the lower triangle of \( \mathbf{A} \):

Matrix at Step \( k \):

\[
\begin{bmatrix}
  l_{21}^{(2)} & a_{22}^{(2)} & a_{23}^{(2)} \\
  l_{31}^{(3)} & a_{32}^{(3)} & a_{33}^{(3)}
\end{bmatrix}
\]
GEM as an $LU$ factorization tool

- We have actually factorized $A$ as

$$A = LU,$$

$L$ 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.
Gauss elimination and LU factorization

GEM in MATLAB

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

```matlab
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
```

A. Donev (Courant Institute)
Pivoting

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

\[ A x = b \]

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

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

Observe \( LU = A \)
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, $PAQ = LU$. 
GEM is a **general** method for **dense matrices** and is commonly used.

Implementing GEM efficiently and stably 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!
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**

$$Ux = y.$$ 

If row pivoting is necessary, the same applies but $L$ or $U$ may be permuted upper/lower triangular matrices,

$$A = \tilde{L}U = (P^T L) U.$$
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.

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.
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!
an =   -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
```
>> [L, U] = lu(A) \% Even better if resolving

L =
    0.1429    1.0000    0.0000
    0.5714    0.5000    1.0000
    1.0000         0    0.0000

U =
     7.0000    8.0000    0.0000
     0.0000    0.8571    3.0000
     0.0000         0    4.5000

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

>> y = L\b;
>> x = U\y; x'
an = -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 \( O(n^2) \) floating-point operations (FLOPs).

- The LU factorization itself costs a lot more, \( O(n^3) \),

\[
\text{FLOPS} \approx \frac{2n^3}{3},
\]

and the triangular solves are negligible for large systems.

- When many linear systems need to be solved with the same \( A \) the factorization can be reused.
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 $x$ and $b$ to be the natural units $x_0$ and $b_0$.
- **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 rescaled matrix $\tilde{A} = D_b^{-1} AD_x$ should have a better conditioning.
- Also note that reordering the variables from most important to least important may also help.
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 $A = \tilde{LU}$.

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