Methods for solving linear systems

Reminders on norms and scalar products of vectors. The application $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ is a norm if

$$1 - \|\underline{v}\| \geq 0, \ \forall \underline{v} \in \mathbb{R}^n \qquad \|\underline{v}\| = 0 \text{ if and only if } \underline{v} = 0;$$
$$2 - \|\alpha \underline{v}\| = |\alpha| \|\underline{v}\| \quad \forall \alpha \in \mathbb{R}, \quad \forall \underline{v} \in \mathbb{R}^n;$$
$$3 - \|\underline{v} + \underline{w}\| \leq \|\underline{v}\| + \|\underline{w}\|, \quad \forall \underline{v}, \underline{w} \in \mathbb{R}^n.$$

Examples of norms of vectors:

$$\|\underline{v}\|_2^2 = \sum_{i=1}^{n} (v_i)^2 \qquad \text{Euclidean norm}$$
$$\|\underline{v}\|_\infty = \max_{1 \leq i \leq n} |v_i| \qquad \text{max norm}$$
$$\|\underline{v}\|_1 = \sum_{i=1}^{n} |v_i| \qquad \text{1-norm}$$

Being in finite dimension, they are all equivalent, with the equivalence constants depending on the dimension n. Ex: $\|v\|_\infty \leq \|v\|_1 \leq n\|v\|_\infty$.

A scalar product is an application $(\cdot,\cdot) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ that verifies:

$1-$ linearity: $(\alpha\underline{v} + \beta\underline{w}, \underline{z}) = \alpha(\underline{v}, \underline{z}) + \beta(\underline{w}, \underline{z})\ \forall \alpha, \beta \in \mathbb{R},\ \forall \underline{v}, \underline{w}, \underline{z} \in \mathbb{R}^n$;

$2-(\underline{v}, \underline{w}) = (\underline{w}, \underline{v})\ \forall \underline{v}, \underline{w} \in \mathbb{R}^n$;

$3-(\underline{v}, \underline{v}) > 0\ \ \forall \underline{v} \neq \underline{0}$ (that is, $(\underline{v}, \underline{v}) \geq 0, (\underline{v}, \underline{v}) = 0$ iff $\underline{v} = \underline{0}$).

To a scalar product we can associate a norm defined as

$$\|\underline{v}\|^2 = (\underline{v}, \underline{v}).$$

Example: $(\underline{v}, \underline{w}) = \sum_{i=1}^{n} v_i w_i, \quad \Longrightarrow\ (\underline{v}, \underline{v}) = \sum_{i=1}^{n} v_i v_i = \|\underline{v}\|_2^2.$

(in this case we can write $(\underline{v}, \underline{w}) = \underline{v} \cdot \underline{w}$ or $\underline{v}^T \underline{w}$ for "column" vectors)

Theorem 1 (Cauchy-Schwarz inequality)

Given a scalar product $(\cdot,\cdot)_*$ *and associated norm* $\|\cdot\|_*$*, the following inequality holds:*

$$|(\underline{v},\underline{w})_*| \leq \|\underline{v}\|_*\|\underline{w}\|_* \qquad \forall v, w \in \mathbb{R}^n$$

Proof.

For $t \in \mathbb{R}$, let $t\underline{v} + \underline{w} \in \mathbb{R}^n$. Clearly, $\|t\underline{v} + \underline{w}\|_* \geq 0$. Hence:

$$\|t\underline{v} + \underline{w}\|_*^2 = t^2\|\underline{v}\|_*^2 + 2t(\underline{v},\underline{w})_* + \|\underline{w}\|_*^2 \geq 0$$

The last expression is a non-negative convex parabola in t (No real roots, or 2 coincident). Then the discriminant is non-positive

$$(\underline{v},\underline{w})_*^2 - \|\underline{v}\|_*^2\|\underline{w}\|_*^2 \leq 0$$

and the proof is concluded. □

Reminders on matrices $A \in \mathbb{R}^{n \times n}$

- A is symmetric if $A = A^T$. The eigenvalues of a symmetric matrix are real.

Reminders on matrices $A \in \mathbb{R}^{n \times n}$

- A is symmetric if $A = A^T$. The eigenvalues of a symmetric matrix are real.
- A symmetric matrix A is positive definite if

$$(A\underline{x}, \underline{x})_2 > 0 \ \forall \underline{x} \in \mathbb{R}^n, \ \underline{x} \neq 0, \quad (A\underline{x}, \underline{x})_2 = 0 \text{ iff } \underline{x} = 0$$

The eigenvalues of a positive definite matrix are positive.

Reminders on matrices $A \in \mathbb{R}^{n \times n}$

- A is symmetric if $A = A^T$. The eigenvalues of a symmetric matrix are real.
- A symmetric matrix A is positive definite if

$$(A\underline{x}, \underline{x})_2 > 0 \ \forall \underline{x} \in \mathbb{R}^n, \ \underline{x} \neq 0, \quad (A\underline{x}, \underline{x})_2 = 0 \text{ iff } \underline{x} = 0$$

  The eigenvalues of a positive definite matrix are positive.
- if A is non singular, $A^T A$ is symmetric and positive definite

Reminders on matrices $A \in \mathbb{R}^{n\times n}$

- A is symmetric if $A = A^T$. The eigenvalues of a symmetric matrix are real.
- A symmetric matrix A is positive definite if

$$(A\underline{x}, \underline{x})_2 > 0 \ \forall \underline{x} \in \mathbb{R}^n, \ \underline{x} \neq 0, \quad (A\underline{x}, \underline{x})_2 = 0 \text{ iff } \underline{x} = 0$$

  The eigenvalues of a positive definite matrix are positive.
- if A is non singular, $A^T A$ is symmetric and positive definite

Proof of the last statment:

- $A^T A$ is always symmetric; indeed $(A^T A)^T = A^T (A^T)^T = A^T A$.

To prove that it is also positive definite we have to show that $(A^T A\underline{x}, \underline{x})_2 > 0 \ \forall \underline{x} \in \mathbb{R}^n, \ \underline{x} \neq 0, \quad (A^T A\underline{x}, \underline{x})_2 = 0$ iff $\underline{x} = 0$. We have:

$$(A^T A\underline{x}, \underline{x})_2 = (A\underline{x}, A\underline{x})_2 = \|A\underline{x}\|_2^2 \geq 0, \text{ and } \|A\underline{x}\|_2^2 = 0 \text{ iff } A\underline{x} = \underline{0}$$

If A is non-singular (i.e., $det(A) \neq 0$), the system $A\underline{x} = \underline{0}$ has only the solution $\underline{x} = \underline{0}$, and this ends the proof.

Norms of matrices

Norms of matrices are applications from $\mathbb{R}^{m\times n}$ to $\mathbb{R}$ satisfying the same properties as for vectors. Among the various norms of matrices we will consider the norms associated to norms of vectors, called natural norms, defined as:

$$|||A||| = \sup_{\underline{v}\neq 0} \frac{\|A\underline{v}\|}{\|\underline{v}\|}$$

It can be checked that this is indeed a norm, that moreover verifies:

$$\|A\underline{v}\| \leq |||A|||\,\|\underline{v}\|, \qquad |||AB||| \leq |||A|||||B|||.$$

Examples of natural norms (of square $n \times n$ matrices)

$$\underline{v} \in \mathbb{R};\quad \|\underline{v}\|_\infty \longrightarrow |||A|||_\infty = \max_{i=1,\cdots,n} \sum_{j=1}^{n} |a_{ij}|,$$

$$\underline{v} \in \mathbb{R};\quad \|\underline{v}\|_1 \longrightarrow |||A|||_1 = \max_{j=1,\cdots,n} \sum_{i=1}^{n} |a_{ij}|,$$

$$\underline{v} \in \mathbb{R};\quad \|\underline{v}\|_2 \longrightarrow |||A|||_2 = \sqrt{|\lambda_{\max}(A^T A)|}.$$

If A is symmetric, $|||A|||_\infty = |||A|||_1$, and $|||A|||_2 = |\lambda_{\text{max in abs val}}(A)|$.
Indeed, if $A = A^T$, then $\max_i \lambda_i(A^T A) = \max_i \lambda_i(A^2) = (\max_i \lambda_i(A))^2$.

The norm $|||A|||_2$ is the *spectral norm*, since it depends on the spectrum of A.

Solving linear systems

The problem: given $\underline{b} \in \mathbb{R}^n$, and $A \in \mathbb{R}^n \times \mathbb{R}^n$, we look for $\underline{x} \in \mathbb{R}^n$ solution of

$$A\underline{x} = \underline{b} \tag{1}$$

Solving linear systems

The problem: given $\underline{b} \in \mathbb{R}^n$, and $A \in \mathbb{R}^n \times \mathbb{R}^n$, we look for $\underline{x} \in \mathbb{R}^n$ solution of

$$A\underline{x} = \underline{b} \tag{1}$$

Problem (1) has a unique solution *if and only if* the matrix A is non-singular (or invertible), i.e., $\exists A^{-1}$ such that $A^{-1}A = AA^{-1} = I$; necessary and sufficient condition for A being invertible is that $det(A) \neq 0$. Then the solution $\underline{x}$ is formally given by $\underline{x} = A^{-1}\underline{b}$.

Solving linear systems

The problem: given $\underline{b} \in \mathbb{R}^n$, and $A \in \mathbb{R}^n \times \mathbb{R}^n$, we look for $\underline{x} \in \mathbb{R}^n$ solution of

$$A\underline{x} = \underline{b} \tag{1}$$

Problem (1) has a unique solution *if and only if* the matrix A is non-singular (or invertible), i.e., $\exists A^{-1}$ such that $A^{-1}A = AA^{-1} = I$; necessary and sufficient condition for A being invertible is that $det(A) \neq 0$. Then the solution $\underline{x}$ is formally given by $\underline{x} = A^{-1}\underline{b}$.

Beware: never invert a matrix unless really necessary, due to the costs, as we shall se later on. (Solving a system with a general full matrix is also expensive, but not nearly as expensive as matrix inversion).

Some example of linear systems

The simplest systems to deal with are diagonal systems:

$$D\underline{x} = \underline{b}$$

$$D = \begin{bmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \cdots & 0 \\ 0 & \cdots & \ddots & 0 \\ 0 & \cdots & & d_{nn} \end{bmatrix} \longrightarrow x_i = \frac{b_i}{d_{ii}} \quad i = 1, 2, \cdots, n$$

The cost in terms of number of operations is negligible, given by n division.

Triangular matrices

Triangular matrices are also easy to handle. If $A = L$ is lower triangular, the system can be solved "forward":

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \longrightarrow \begin{cases} x_1 = \dfrac{b_1}{l_{11}} \\ x_2 = \dfrac{b_2 - l_{21}x_1}{l_{22}} \\ \vdots \\ x_n = \dfrac{b_n - \sum_{j=1}^{n-1} l_{nj}x_j}{l_{nn}} \end{cases}$$

Counting the operations: for x_1 we have 1 product and 0 sums; for x_2 2 products and 1 sum, ..., and for x_n n products and $n-1$ sums, that is, $1+2+\cdot+n = \dfrac{n(n+1)}{2}$ products, plus $1+2+\cdots+n-1 = \dfrac{(n-1)n}{2}$ sums, for a total number of operations $= n^2$.

A special case of lower triangular matrix (useful later...)

Assume L has only 1 on the diagonal, e.g., $l_{ii} = 1$:

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix} \longrightarrow \begin{cases} x_1 = b_1 \\ x_2 = b_2 - l_{21}x_1 \\ \vdots \\ x_n = b_n - \sum_{j=1}^{n-1} l_{nj}x_j \end{cases}$$

Then the following two algorithms, for solving $L\underline{x} = \underline{b}$ are equivalent:

forward substitution as above

Input: $L \in \mathbb{R}^{n\times n}$, lower t., and $b \in \mathbb{R}^n$
for $i = 2, \ldots, n$
  for $j = 1, \ldots, i-1$
    $b_i = b_i - l_{i,j}b_j$
  end
end
the solution is in $\underline{b}$, i.e., $\underline{x} \leftarrow \underline{b}$

equivalent to forward substitution

Input: $L \in \mathbb{R}^{n\times n}$, lower t., and $b \in \mathbb{R}^n$
for $k = 1, \ldots, n-1$
  for $i = k+1, \ldots, n$
    $b_i = b_i - l_{i,k}b_k$
  end
end
the solution is in $\underline{b}$, i.e., $\underline{x} \leftarrow \underline{b}$

Triangular matrices

Upper triangular systems (if $A = U$ is upper triangular) are also easy to deal with, and can be solved "backward" with the same costs as lower triangular systems:

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & & \cdots & u_{nn} \end{bmatrix} \longrightarrow \begin{cases} x_n = \dfrac{b_n}{u_{nn}} \\ x_{n-1} = \dfrac{b_{n-1} - u_{n-1n}x_n}{u_{n-1n-1}} \\ \vdots \\ x_1 = \dfrac{b_1 - \sum_{j=2}^{n} u_{1j}x_j}{u_{11}} \end{cases}$$

Homework

Write the above algorithm in MATLAB

Costs for solving triangular systems

We saw that for solving the system we perform n^2 operations.

To have an idea of the time necessary to solve a triangular system, suppose that the number of equations is $n = 100.000$, and the computer performance is a TERAFLOP $= 10^{12} FLOPS$ (floating-point operations per second); the time in seconds is given by

$$t = \frac{\#operations}{\#flops} = \frac{10^{10} ops}{10^{12} flops} = \frac{1}{100} sec.$$

(pretty quick)

The next target of High Performance Computing is EXAFLOP $= 10^{18} FLOPS$

General matrices

For a general full matrix the cost can become extremely high if we do not work properly.

For example, Cramer method is awfully expensive, even for small matrices:

$$x_i = \frac{det(A_i)}{det(A)}, \quad A_i \text{ has } \underline{b} \text{ in the i}^{th} \text{ column}$$

the number of operations for computing the determinant of an $n \times n$ matrix is of the order on $n!$ (I mean factorial of n). We have to compute $n+1$ determinants, so the total number of operations is of the order of $(n+1)!$. Try to solve a 20×20 system on a computer with power $10^{12} flops$: how much do you have to wait?

Numerical methods

There are two classes of methods for solving the linear system (1): direct and iterative.

Direct methods: they give the exact solution (up to computer precision) in a finite number of operations.

Iterative methods: starting from an initial guess $\underline{x}^{(0)}$ they construct a sequence $\{\underline{x}^{(k)}\}$ such that

$$\underline{x} = \lim_{k \to \infty} \underline{x}^{(k)}.$$

Direct methods

The most important direct method is GEM (Gaussian elimination method).

With GEM, the original system (1) is transformed into an equivalent system (i.e., having the same solution)

$$U\underline{x} = \widetilde{\underline{b}} \qquad (*)$$

with U upper triangular matrix. This is done in $n-1$ steps where, at each step, one column is eliminated, that is, all the coefficients of that column below the diagonal are transformed into zeros. The cost for computing U is of the order of $\frac{2}{3}n^3$, see later... the cost for computing $\widetilde{b}$ is of the order of n^2; the cost for solving the upper triangular system is n^2, so that the total cost is $\sim \frac{2}{3}n^3 + 2n^2 \sim \frac{2}{3}n^3$.

GEM algorithm

$A^{(1)} := A,\ b^{(1)} := b$

$$A^{(1)}x = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} = b^{(1)}$$

GEM algorithm

$A^{(1)} := A,\ b^{(1)} := b$

$$A^{(1)}x = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} = b^{(1)}$$

- We eliminate the first column:

$$A^{(2)}x = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \dots & a_{nn}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix} = b^{(2)}$$

$$l_{i1} = a_{i1}^{(1)}/a_{11}^{(1)}, \quad a_{ij}^{(2)} = a_{ij}^{(1)} - l_{i1}a_{1j}^{(1)}, \quad b_i^{(2)} = b_i^{(1)} - l_{i1}b_1^{(1)}$$

GEM algorithm

- We go on eliminating columns in the same way. At the $k-$th step:

$$A^{(k)}x = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & & \vdots \\ 0 & \dots & 0 & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix} = b^{(k)}$$

GEM algorithm

- We go on eliminating columns in the same way. At the $k-$th step:

$$A^{(k)}x = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & & \vdots \\ 0 & \dots & 0 & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_k^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix} = b^{(k)}$$

- At the last step $A^{(n)}x = b^{(n)}$ with U upper triangular,

$$\Longrightarrow U := A^{(n)},\ \widetilde{b} := b^{(n)}$$

GEM pseudocode

Input: $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$
for $k = 1, \ldots, n-1$
  for $i = k+1, \ldots, n$
    $l_{i,k} = a_{i,k} / a_{k,k}$
    $a_{i,k:n} = a_{i,k:n} - l_{i,k} a_{k,k:n}$
    $b_i = b_i - l_{i,k} b_k$
  end
end
define $U = A$, $\widetilde{b} = b$, then solve $Ux = \widetilde{b}$ with back substitution

remark: we do not need to define U and $\widetilde{b}$, it is just to be consistent with the notation of the previous slides

Homework

Write the above algorithm in MATLAB

Possible troubles and remedy

The condition $det(A) \neq 0$ is not sufficient to guarantee that the elimination procedure will be successful. For example $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Possible troubles and remedy

The condition $det(A) \neq 0$ is not sufficient to guarantee that the elimination procedure will be successful. For example $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

To avoid this the remedy is the "pivoting" algorithm:

- first step: before eliminating the first column, look for the coefficient of the column biggest in absolute value, the so-called "pivot"; if r is the row where the pivot is found, exchange the first and the r^{th} row.

Possible troubles and remedy

The condition $det(A) \neq 0$ is not sufficient to guarantee that the elimination procedure will be successful. For example $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

To avoid this the remedy is the "pivoting" algorithm:

- first step: before eliminating the first column, look for the coefficient of the column biggest in absolute value, the so-called "pivot"; if r is the row where the pivot is found, exchange the first and the r^{th} row.
- second step: before eliminating the second column, look for the coefficient of the column biggest in absolute value, starting from the second row; if r is the row where the pivot is found, exchange the second and the r^{th} row.

Possible troubles and remedy

The condition $det(A) \neq 0$ is not sufficient to guarantee that the elimination procedure will be successful. For example $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

To avoid this the remedy is the "pivoting" algorithm:

- first step: before eliminating the first column, look for the coefficient of the column biggest in absolute value, the so-called "pivot"; if r is the row where the pivot is found, exchange the first and the r^{th} row.
- second step: before eliminating the second column, look for the coefficient of the column biggest in absolute value, starting from the second row; if r is the row where the pivot is found, exchange the second and the r^{th} row.

$\vdots$

- step j: before eliminating the column j, look for the pivot in this column, from the diagonal coefficient down to the last row. If the pivot is found in the row r, exchange the rows j and r.

This is the pivoting procedure on the rows, which amounts to multiply at the left the matrix A by a permutation matrix P. (An analogous procedure can be applied on the columns, or globally).
The pivoting procedure corresponds then to solve, instead of (1), the system

$$PA\underline{x} = P\underline{b} \tag{2}$$

GEM pseudocode with pivoting

Input as before: $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$
for $k = 1, \ldots, n-1$
    select $j \geq k$ that maximise $|a_{jk}|$
    $a_{j,k:n} \longleftrightarrow a_{k,k:n}$
    $b_j \longleftrightarrow b_k$
    for $i = k+1, \ldots, n$
        $l_{i,k} = a_{i,k}/a_{k,k}$
        $a_{i,k:n} = a_{i,k:n} - l_{i,k} a_{k,k:n}$
        $b_i = b_i - l_{i,k} b_k$
    end
end
define $U = A$, $\widetilde{b} = b$, then solve $Ux = \widetilde{b}$ with back substitution

remark: we do not need to define U and $\widetilde{b}$, it is just to be consistent with the notation of the previous slides

LU factorisation, consists in looking for two matrices L lower triangular, and U upper triangular, both non-singular, such that

$$LU = A \tag{3}$$

If we find these matrices, system (1) splits into two triangular systems easy to solve:

$$A\underline{x} = \underline{b} \;\rightarrow\; L(U\underline{x}) = \underline{b} \;\rightarrow\; \begin{cases} L\underline{y} = \underline{b} & \text{solved forward} \\ U\underline{x} = \underline{y} & \text{solved backward} \end{cases}$$

Mathematically equivalent to GEM: matrix U is the same, $\widetilde{b} = L^{-1}b$.

LU: unicity of the factors

The factorization $LU = A$ is unique?

$$\underbrace{\begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & & \cdots & u_{nn} \end{pmatrix}}_{U} = \underbrace{\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}}_{A}$$

The unknowns are the coefficients l_{ij} of L, which are $1 + 2 + \cdot + n = \frac{n(n+1)}{2}$, and the coefficients u_{ij} of U, also $\frac{n(n+1)}{2}$, for a total of $n^2 + n$ unknowns.
We only have n^2 equations (as many as the number of coefficients of A), so we need to fix n unknowns. Usually, the diagonal coefficients of L are set equal to 1: $l_{ii} = 1$. If you do so...

GEM vs LU (pseudocodes)

GEM

Input: $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$
for $k = 1, \ldots, n-1$
  for $i = k+1, \ldots, n$
    $l_{i,k} = a_{i,k}/a_{k,k}$
    $a_{i,k:n} = a_{i,k:n} - l_{i,k}a_{k,k:n}$
    $b_i = b_i - l_{i,k}b_k$
  end
end
set $U = A$, then solve $Ux = b$ with back substitution

- we can store the $l_{i,k}$in a matrix:

$$L = \begin{bmatrix} ? & ? & \cdots & ? \\ l_{2,1} & ? & \cdots & ? \\ \vdots & \vdots & & \vdots \\ l_{n,1} & l_{n,2} & \cdots & ? \end{bmatrix}$$

that can be completed as a lower triangular matrix

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ l_{n,1} & l_{n,2} & \cdots & 1 \end{bmatrix}$$

- the loops on the left replace $\underline{b}$ by $L^{-1}\underline{b}$ (see the "equivalent forward substitution" algoritms): $\underline{b} \leftarrow L^{-1}\underline{b}$
- similarly $A \leftarrow L^{-1}A$, which is called U and is an upper triangular matrix:

$$L^{-1}A = U \Rightarrow A = LU$$

GEM vs LU (pseudocodes)

GEM

Input: $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$
for $k = 1, \ldots, n-1$
  for $i = k+1, \ldots, n$
    $l_{i,k} = a_{i,k}/a_{k,k}$
    $a_{i,k:n} = a_{i,k:n} - l_{i,k}a_{k,k:n}$
    $b_i = b_i - l_{i,k}b_k$
  end
end
set $U = A$, then solve $Ux = b$ with back substitution

LU

Input: $A \in \mathbb{R}^{n\times n}$
$L = I_n \in \mathbb{R}^{n\times n}$
for $k = 1, \ldots, n-1$
  for $i = k+1, \ldots, n$
    $l_{i,k} = a_{i,k}/a_{k,k}$
    $a_{i,k:n} = a_{i,k:n} - l_{i,k}a_{k,k:n}$
  end
end
Set $U = A$ and output: U and L

GEM vs LU (pseudocodes)

GEM

Input: $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$
for $k = 1, \ldots, n-1$
  for $i = k+1, \ldots, n$
    $l_{i,k} = a_{i,k}/a_{k,k}$
    $a_{i,k:n} = a_{i,k:n} - l_{i,k}a_{k,k:n}$
    $b_i = b_i - l_{i,k}b_k$
  end
end
set $U = A$, then solve $Ux = b$ with back substitution

LU

Input: $A \in \mathbb{R}^{n\times n}$
$L = I_n \in \mathbb{R}^{n\times n}$
for $k = 1, \ldots, n-1$
  for $i = k+1, \ldots, n$
    $l_{i,k} = a_{i,k}/a_{k,k}$
    $a_{i,k:n} = a_{i,k:n} - l_{i,k}a_{k,k:n}$
  end
end
Set $U = A$ and output: U and L

Homework

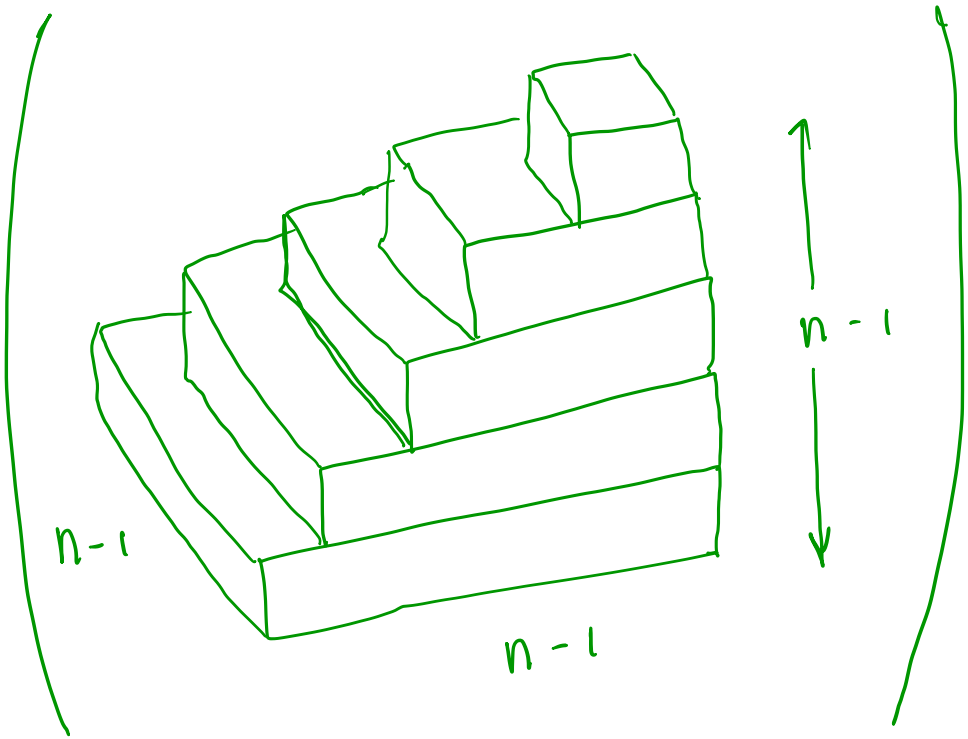
Write the LU algorithm in MATLAB as in the previous slide, then add pivoting in order to return L,U and the permutation matrix P

computation cost of LU (GEM is similar)

when $k=1$ the operations are $(n-1)\left(1+2(n-1)+2\right) \simeq 2(n-1)^2$

" $k=2$ " " " " $(n-2)\left(1+2(n-2)+2\right) \simeq 2(n-2)^2$

$\vdots$ $\vdots$

volume of 

$\simeq \frac{1}{3}(n-1)^3$

total FLOPS $= 2 \cdot \frac{1}{3}(n-1)^3 = \frac{2}{3}n^3 + \cdots \simeq \frac{2}{3}n^3$

LU-continued

GEM and LU have the same computational cost: $\frac{2}{3}n^3$

In GEM, the coefficients l_{ik} are discarded after application to the right-hand side b, while in the LU factorisation they are stored in the matrix L.
If we have to solve a single linear system, GEM is preferable (less memory storage).
If we have to solve many systems with the same matrix and different right-hand sides LU is preferable (the heavy cost is payed only once).

Pivoting is applied also to the LU factorization to ensure that the factorisation is succesful

$$PA = LU \quad \Longrightarrow \quad PAx = LUx = Pb$$

The Matlab function that computes L and U is $lu(.,.)$.

LU versus GEM

If one needs to compute the inverse of a matrix, LU is the cheapest way. Indeed, recalling the definition, the inverse of a matrix A is the matrix A^{-1} solution of

$$AA^{-1} = I$$

LU versus GEM

If one needs to compute the inverse of a matrix, LU is the cheapest way. Indeed, recalling the definition, the inverse of a matrix A is the matrix A^{-1} solution of

$$AA^{-1} = I$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} \underbrace{c_{11}}_{\underline{c}^{(1)}} & \underbrace{c_{12}}_{\underline{c}^{(2)}} & \cdots & \underbrace{c_{1n}}_{\underline{c}^{(n)}} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} \underbrace{1}_{\underline{e}^{1}} & \underbrace{0}_{\underline{e}^{2}} & \cdots & \underbrace{0}_{\underline{e}^{n}} \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

LU versus GEM

If one needs to compute the inverse of a matrix, LU is the cheapest way. Indeed, recalling the definition, the inverse of a matrix A is the matrix A^{-1} solution of

$$AA^{-1} = I$$

Hence, each column $\underline{c}^{(i)}$ of A^{-1} is the solution of

$$A\underline{c}^{(i)} = \underline{e}^{(i)}, \qquad i = 1, 2, \cdots, n$$

with $\underline{e}^{(i)} = (0, 0, \cdots, 1, \cdots, 0)$. The factorisation can be done once and for all at the cost of $O(2n^3/3)$ operations; for each column we have to solve 2 triangular systems ($2n^2$ operations) so that the total cost is of the order of $\frac{2}{3}n^3 + n \times 2n^2 = \frac{8}{3}n^3$.
In case of pivoting, we solve $PA\underline{c}^{(i)} = P\underline{e}^{(i)} = P_{:,i}$, $i = 1, 2, \cdots, n$.

Computation of the determinant

We can use the LU factorisation to compute the determinant of a matrix. Indeed, if $A = LU$, thanks to the Binet theorem we have

$$\det(A) = \det(L)\det(U) = \prod_{i=1}^{n} l_{ii} \prod_{i=1}^{n} u_{ii} = \prod_{i=1}^{n} u_{ii}$$

Thus the cost to compute the determinant is the same of the LU factorisation.

Computation of the determinant

We can use the LU factorisation to compute the determinant of a matrix. Indeed, if $A = LU$, thanks to the Binet theorem we have

$$\det(A) = \det(L)\det(U) = \prod_{i=1}^{n} l_{ii} \prod_{i=1}^{n} u_{ii} = \prod_{i=1}^{n} u_{ii}$$

Thus the cost to compute the determinant is the same of the LU factorisation.
In the case of pivoting, $PA = LU$ and then

$$\det(A) = \frac{\det(L)\det(U)}{\det(P)} = \frac{\det(U)}{\det(P)}$$

It turns out that $\det(P) = (-1)^{\delta}$ where $\delta = \#$ of row exchanges in the LU factorisation.
Matlab function: det(·)

Cholesky factorization

If A is symmetric ($A = A^T$) and positive definite (positive eigenvalues) a variant of LU is due to Cholesky: there exists a non-singular lower triangular matrix L such that

$$LL^T = A$$

Costs: approximately $\sim \frac{n^3}{3}$ (half the cost of LU, using the symmetry of A).

Matlab function: chol(.,.)

Sparse Matrices

"A matrix is sparse if many of its coefficients are zero. The interest in sparsity arises because its exploitation can lead to enormous computational savings and because many large matrix problems that occur in practice are sparse."

- Page 1, Direct Methods for Sparse Matrices, 2nd Edition, 2017.

Sparse Matrices

"A matrix is sparse if many of its coefficients are zero. The interest in sparsity arises because its exploitation can lead to enormous computational savings and because many large matrix problems that occur in practice are sparse."

- Page 1, Direct Methods for Sparse Matrices, 2nd Edition, 2017.

The **sparsity** of an $n \times n$ matrix A is

$$\frac{\text{nnz}(A)}{n^2}$$

where $\text{nnz}(A) = \#$ of nonzero entries in A. A matrix is **sparse** if its sparsity is $\ll 1$. A matrix that is not sparse is **dense**.

Sparse matrices

- Sparse matrices are extremely common in engineering and computer science. Some examples:
  - Network theory (e.g. social networks).
  - Data analysis and machine learning.
  - Discretization of differential equations.
  - ...

Sparse matrices

- Sparse matrices are extremely common in engineering and computer science. Some examples:
  - Network theory (e.g. social networks).
  - Data analysis and machine learning.
  - Discretization of differential equations.
  - ...
- Sparse matrices represent problems with possibly a large of number of variables, but where each variable "interacts" directly with few other variables (local interactions).

Formats of sparse matrices

- To save memory, it is convenient to store only the nonzero entries of a sparse matrix. There are several data structure that allow this.
- Compressed Sparse Column (CSC). The matrix A is specified by three arrays: `val`, `row_ind` and `col_ptr`.
  - `val` stores the nonzero entries of A, ordered from from top to bottom and left to right.
  - `row_ind` stores the row indices of the nonzero entries of A.
  - `col_ptr` stores the indices of the elements in `val` which start a column of A.

  This is the format used by Matlab.
- Example:

$$A = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 7 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 8 & 2 \end{bmatrix}$$

$$\begin{aligned} \texttt{val} &= \begin{bmatrix} 4 & 7 & 3 & 3 & 8 & 2 \end{bmatrix} \\ \texttt{row\_ind} &= \begin{bmatrix} 1 & 2 & 2 & 3 & 4 & 4 \end{bmatrix} \\ \texttt{col\_ptr} &= \begin{bmatrix} 1 & 3 & 4 & 6 \end{bmatrix} \end{aligned}$$

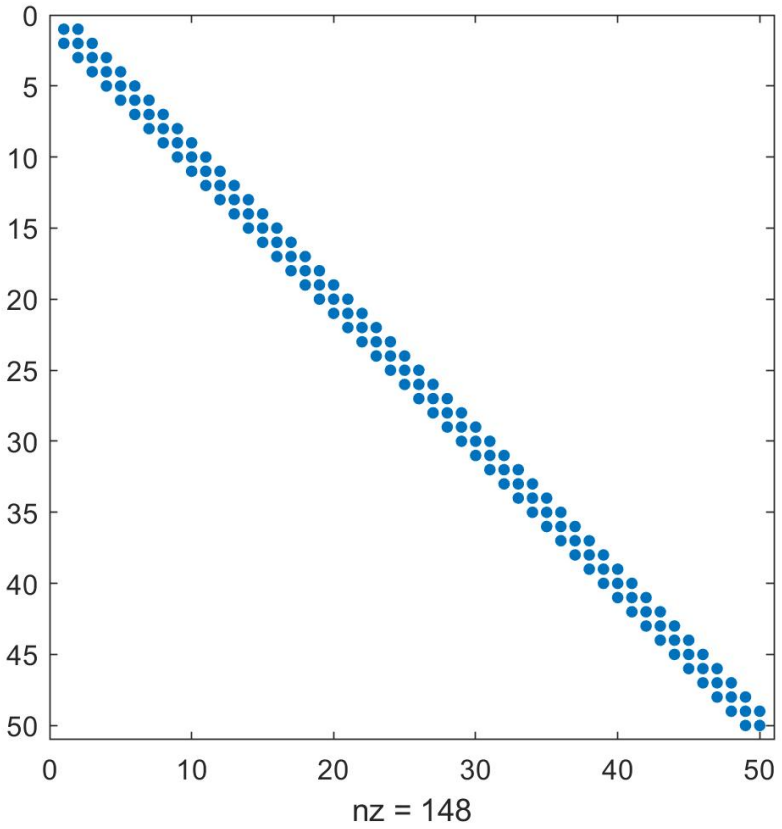
Sparsity and direct solvers

- We can take advantage of sparsity in Gaussian elimination, performing only the operations that are necessary (e.g. only the nonzero entries below a pivot are eliminated, and when summing two rows only the nonzero entries are summed). This allows to beat the $O(n^3)$ cost for dense matrices.
- This works particularly well in case of **banded matrices** (the nonzero are concentrated in a narrow "band" around the diagonal). In this case, a system can be solved in $O(n)$ operations.

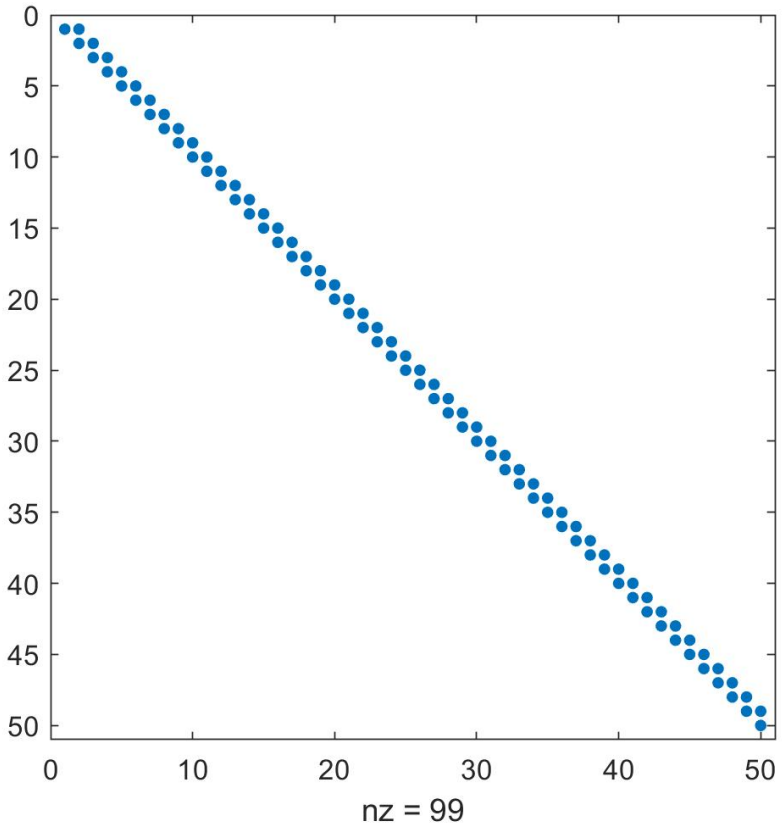
Example of a sparse matrix

A from 1D Poisson problem on $[0, 1]$ (discretized with FEM/FD, $h = 0.02$):

sparsity pattern of A



sparsity pattern of U



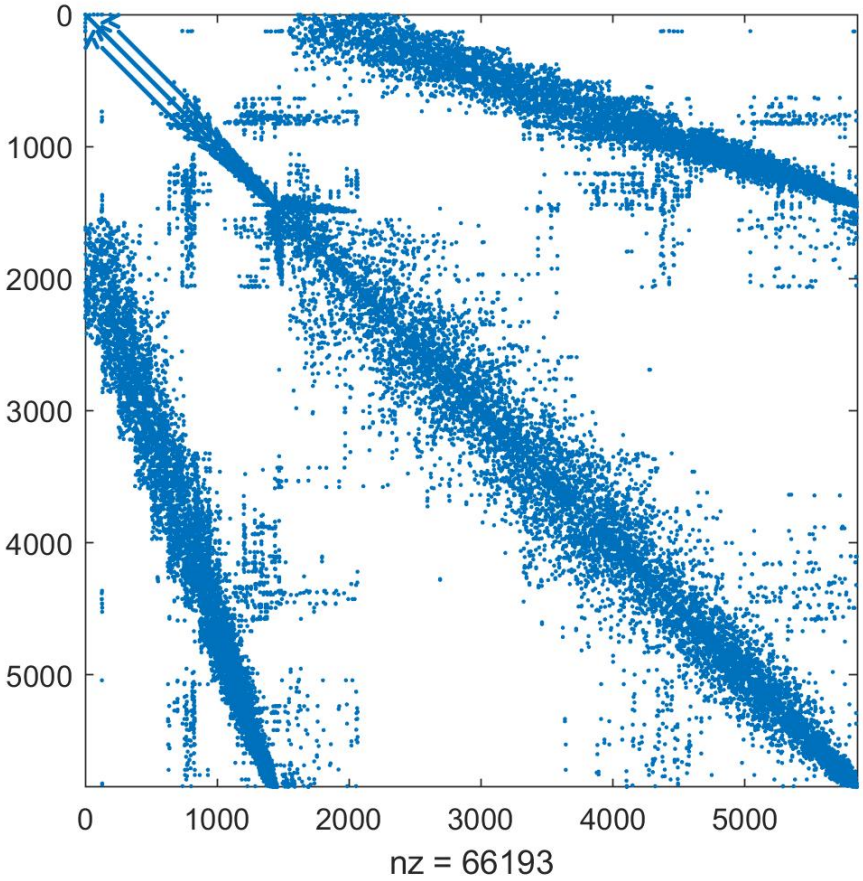
Fill-in

In general, the factor U (and L) can have much more nonzero entries than A. This phenomenon, known as **fill-in** significantly increases time and memory consumption, and represents the main drawback of direct solvers for sparse systems.

Example of fill-in

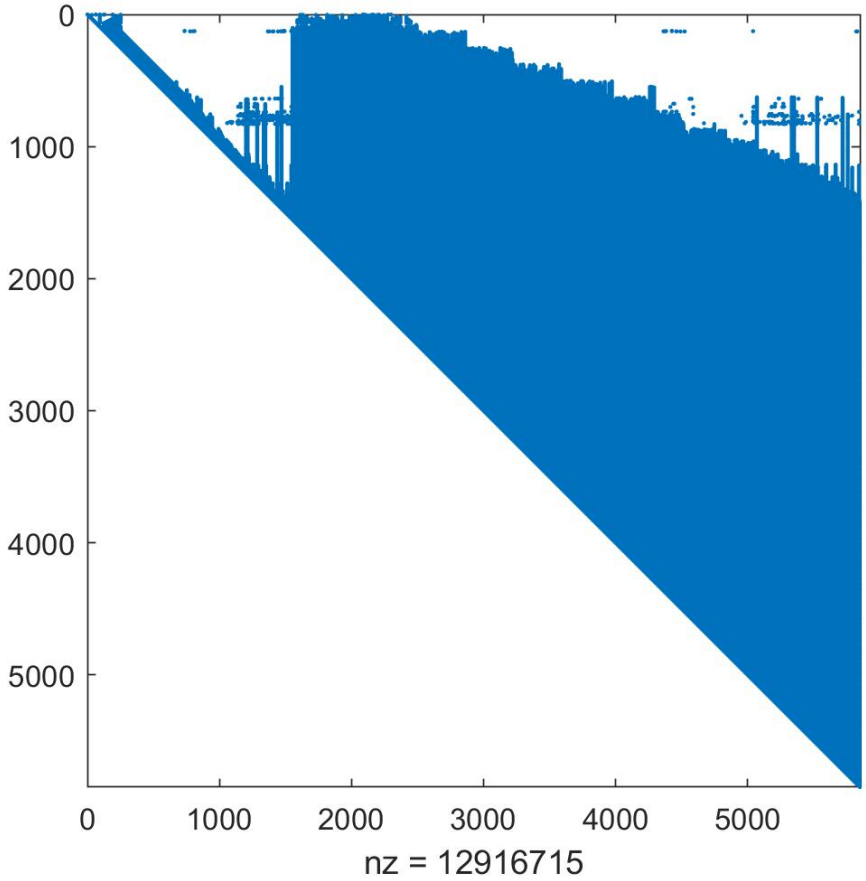
We consider a matrix A generated by a Finite Element Method for solving a 2D Poisson problem on the unit circle (meshsize $h = 0.05$).

sparsity pattern of A



memory(A) $\approx$ 1 MB

sparsity pattern of U



memory(U) $\approx$ 242 MB

Orderings

- The level of fill-in is often sensitive to the ordering of the variables
- Example:

$$A = \begin{bmatrix} x & x & x & x & x \\ x & x & & & \\ x & & x & & \\ x & & & x & \\ x & & & & x \end{bmatrix} \quad \Longrightarrow \quad U = \begin{bmatrix} x & x & x & x & x \\ & x & x & x & x \\ & & x & x & x \\ & & & x & x \\ & & & & x \end{bmatrix}$$

  A is sparse but U is completely dense.

- But if we re-order rows and columns from last to first:

$$A = \begin{bmatrix} x & & & & x \\ & x & & & x \\ & & x & & x \\ & & & x & x \\ x & x & x & x & x \end{bmatrix} \quad \Longrightarrow \quad U = \begin{bmatrix} x & & & & x \\ & x & & & x \\ & & x & & x \\ & & & x & x \\ & & & & x \end{bmatrix}$$

  In this case, there is no fill-in.

Appendix on pivoting for the LU algorithm

Pivoting in GEM and LU factorization

P is a permutation matrix if it has only one entry 1 on each row and each column, and then only ϕ.

It produces permutation of rows when multiplying on the left and of columns when multiplying on the right. For example

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \qquad A = \begin{bmatrix} \overline{\quad r_1 \quad} \\ \overline{\quad r_2 \quad} \\ \overline{\quad r_3 \quad} \\ \overline{\quad r_4 \quad} \end{bmatrix} = \left[\begin{array}{c|c|c|c} c_1 & c_2 & c_3 & c_4 \end{array}\right]$$

$$PA = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 \\ \hline r_2 \\ \hline r_3 \\ \hline r_4 \end{bmatrix} = \begin{bmatrix} r_3 \\ \hline r_1 \\ \hline r_2 \\ \hline r_4 \end{bmatrix}$$

$$AP = \left[\begin{array}{c|c|c|c} c_1 & c_2 & c_3 & c_4 \end{array}\right] \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c|c|c} c_2 & c_3 & c_1 & c_4 \end{array}\right]$$

Successive row permutation

$$P_n P_{n-1} \cdots P_2 P_1 A = \begin{cases} \left(P_n \left(P_{n-1} \cdots \left(P_2 \left(P_1 A\right)\right)\cdots\right)\right) \\ \underbrace{\left(P_n P_{n-1} \cdots P_2 P_1\right)}_{P} A \end{cases}$$

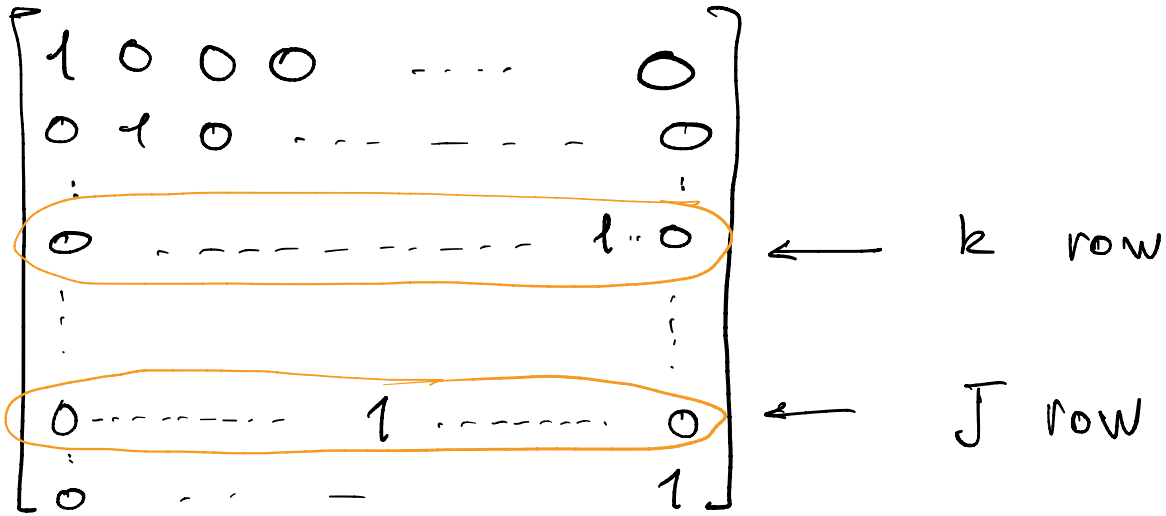
GEM with (partial) pivoting looks like

$$A^{(1)} = A$$
$$A^{(2)} = L^{(1)} P^{(1)} A^{(1)}$$
$$A^{(3)} = L^{(2)} P^{(2)} A^{(2)}$$
$$\vdots$$
$$A^{(n)} = L^{(n-1)} P^{(n-1)} A^{(n-1)}$$

$P^{(k)}$ = switch row k with row $j \geq k$ when left multiplying

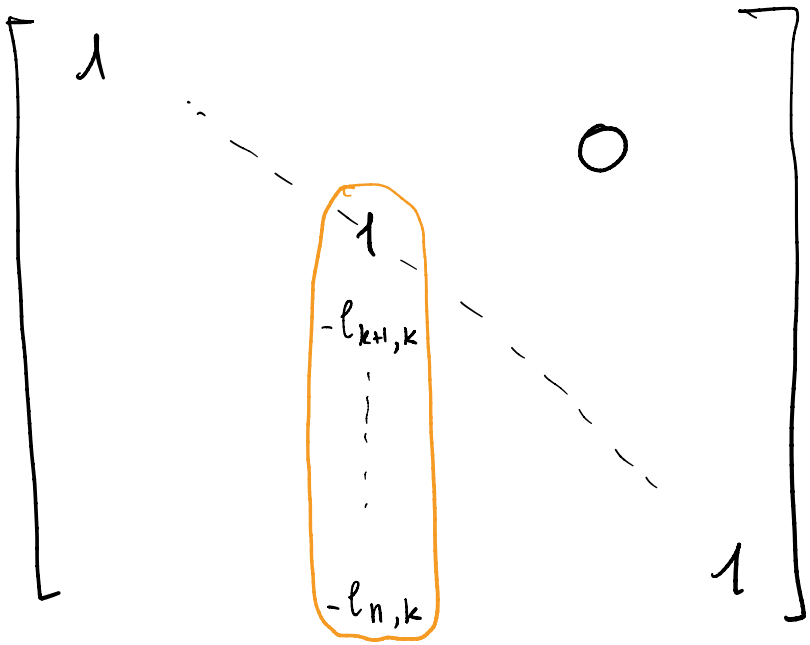
$L^{(k)}$ = eliminate the underdiagonal entries when multiplying from the left

$P^{(k)}$ is a permutation matrix



$L^{(k)}$ is called an atomic lower triangular matrix

$L^{(k)} =$



k column

Assuming we do not have pivoting:

$$A^{(n)} = L^{(n-1)} \cdots L^{(2)} L^{(1)} A$$

$$\left(L^{(1)}\right)^{-1} \cdots \left(L^{(n-1)}\right)^{-1} A^{(n)} = A$$

where $A^{(n)}$ is upper triangular $= U$

$\left(L^{(1)}\right)^{-1} \cdots \left(L^{(n-1)}\right)^{-1}$ = lower triang $= L$

and easy to compute ...

$$L = \begin{pmatrix} 1 & & 0 \\ \ell_{2,1} & \ddots & \\ \vdots & & \\ \ell_{n,1} & \ell_{n,n-1} & 1 \end{pmatrix}$$

but pivoting mix everything:

$$A^{(1)} = A$$
$$A^{(2)} = L^{(1)} P^{(1)} A^{(1)}$$
$$A^{(3)} = L^{(2)} P^{(2)} A^{(2)}$$
$$\vdots$$
$$A^{(n)} = L^{(n-1)} P^{(n-1)} A^{(n-1)}$$

therefore:

$$U = A^{(n)} = L^{(n-1)} P^{(n-1)} \cdots L^{(1)} P^{(1)} A$$
$$\neq L^{(n-1)} \cdots L^{(1)} \cdot P^{(n-1)} \cdots P^{(1)} A$$

We cannot commute the matrices! Things are more complicated and are based on the following: if

$$P^{(j)} L^{(k)} = \tilde{L}^{(k)} P^{(J)}$$

where $P^{(J)}$ switch J with $i \geq J$
$J, i \geq k$

$\tilde{L}^{(k)}$ = obtained from $L^{(k)}$ by switching $L^{(k)}_{i,k} \longleftrightarrow L^{(k)}_{J,k}$

this observation allows us to rearrange the factors

$$U = A^{(n)} = L^{(n-1)} P^{(n-1)} L^{(n-2)} P^{(n-2)} \cdots A$$

$$= L^{(n-1)} \tilde{L}^{(n-2)} P^{(n-1)} P^{(n-2)} \cdots A$$

and so on ...

$$= \underbrace{L^{(n-1)} \tilde{L}^{(n-2)} \tilde{\tilde{L}}^{(n-3)} \cdots \tilde{\tilde{\tilde{L}}}^{(1)}}_{\text{to be inverted}} \cdot \underbrace{P^{(n-1)} \cdots P^{(1)}}_{P} A$$

but still lower tr. and atomic

and finally :

$$\left(\tilde{\tilde{L}}^{(1)}\right)^{-1} \cdot \ldots \cdot \left(\tilde{L}^{(n-2)}\right)^{-1} \left(L^{(n-1)}\right)^{-1} U = P A$$

$$\tilde{L} U = P A$$

which is named "Doolittle decomposition"

$\tilde{L}$ is easy to compute, as is $L = (L^{(1)})^{-1} \cdot \ldots \cdot \left(L^{(n-1)}\right)^{-1}$

$\tilde{L}$ entries are the same of L but rearranged

In fact $\tilde{L}$ is computed on the fly in $\longrightarrow$

final algorithm for LU = PA factorization →

```
1  function [L,U,P]=LU_pivot(A)
2  % LU factorization with partial (row) pivoting
3  % K. Ming Leung, 02/05/03
4
5  [n,n]=size(A);
6  L=eye(n); P=L; U=A;
7  for k=1:n
8      [pivot m]=max(abs(U(k:n,k)));
9      m=m+k-1;
10     if m~=k
11         % interchange rows m and k in U
12         temp=U(k,:);
13         U(k,:)=U(m,:);
14         U(m,:)=temp;
15         % interchange rows m and k in P
16         temp=P(k,:);
17         P(k,:)=P(m,:);
18         P(m,:)=temp;
19         if k >= 2
20             temp=L(k,1:k-1);
21             L(k,1:k-1)=L(m,1:k-1);
22             L(m,1:k-1)=temp;
23         end
24     end
25     for j=k+1:n
26         L(j,k)=U(j,k)/U(k,k);
27         U(j,:)=U(j,:)-L(j,k)*U(k,:);
28     end
29 end
```

this is the needed rearranging in the L matrix