Systems of linear equations

Given a system of equations with dimension *n* x *n:*

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ \ldots &= \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

it can be written in matrix form as:

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

where A is the matrix of the coefficients, **x** are the unknowns and **b** is the vector of known terms.

Why a numerical solution?

- The problem is always analytically solvable if the matrix is invertible (namely if $|A| \neq 0$)
- Often, the numerical solution of complex problems (e.g., differential equations) can be re-formulated as a system of linear algebraic equations with very large *N*!
- Problem: find an algorithm which computes the numerical solution with the highest possible precision by using the lowest number of operations.

Solution through the Cramer's rule!

- The Cramer's rule yields the solution:

$$x_i = \frac{|A_i|}{|A|} \qquad i = 1, \ldots, n$$

  where $|A_i|$ is the determinant of the matrix obtained by substituting the column-vector **b** to the *i*-th column of *A.*

- How complex is this algorithm? We keep into account only multiplications and divisions, by neglecting additions!

Solution through the Cramer’s rule!

- The determinant of a *n*x*n* matrix can be defined as:

$$|A| = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$$

where S_n is the set of all permutations of the first *n* integer numbers, σ is a generic permutation of such elements, $\sigma(i)$ is the *i*-th figure of this permutation and sign(σ) indicates the sign of the permutation (+1 for even permutations, -1 for odd).

Solution through the Cramer's rule!

- Example for n=3:

  S_n has $n!$ elements = 6

  even permutations: 123, 231, 312;

  odd permutations: 213, 132, 321;

$$|A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} =$$

$$= a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} +$$

$$- a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32} - a_{13}a_{22}a_{31}$$

Solution through the Cramer's rule!

- Therefore, each $n \times n$ determinant requires:

  $n!$ times $n+1$ products = $(n+1) \times n!$

  We have to compute n unknowns x_i, that is:

  - n determinants for the numerator
  - 1 determinant (equal for all i) for the denominator

  for a total of $n+1$ determinants, namely:

  $(n+1)^2 \times n!$ operations.

  This is a huge number for large n!!!

Triangular systems

- A triangular system is such that:

$$a_{ij} = 0,\ \forall j > i \quad \Rightarrow$$ lower triangular

$$a_{ij} = 0,\ \forall j < i \quad \Rightarrow$$ upper triangular

- For instance:

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad \text{is LT}$$

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad \text{is UT}$$

Forward and Backward substitutions

- The solution of LT system is trivial (if $l_{ii} \neq 0$) with the forward substitutions algorithm:

$$x_1 = b_1/l_{11}; \qquad x_2 = \frac{b_2 - l_{21}x_1}{l_{22}}$$

$$x_3 = \frac{b_3 - l_{32}x_2 - l_{31}x_1}{l_{33}}$$

- For the second system we can use the backward substitutions algorithm (if $u_{ii} \neq 0$):

$$x_3 = b_3/u_{33}; \qquad x_2 = \frac{b_2 - u_{23}x_3}{u_{22}}$$

$$x_1 = \frac{b_1 - u_{12}x_2 - u_{13}x_3}{u_{11}}$$

Generic triangular systems

- The two algorithms can be easily generalized to the *n x n* case:

$$\begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$\begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Generic triangular systems

- For the FS:

$$x_1 = \frac{b_1}{l_{11}}; \qquad x_i = \frac{1}{l_{ii}}\left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right) \quad i = 2, \ldots, n$$

- For the BS:

$$x_n = \frac{b_n}{u_{nn}}; \qquad x_i = \frac{1}{u_{ii}}\left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j\right) \quad i = n-1, \ldots, 1$$

Complexity of FS and BS

- For the FS:
  - Computation of x_1 requires 1 product;
  - Computation of x_2 requires 2 products;
  - ...
  - Computation of x_i requires i products;
- The total complexity is:

$$\boxed{1 + 2 + \ldots + n = \frac{n(n+1)}{2} \sim n^2}$$

- Proof:

$$\left.\begin{array}{r} s = 1 + 2 + 3 + \ldots + n \\ s = n + (n-1) + (n-2) + \ldots + 1 \end{array}\right\} \Rightarrow 2s = (n+1) + (n-1+2) + (n-2+3) + \ldots + (1+n)$$

Gaussian elimination

- The case of triangular matrices is a very special one, however a generic matrix can be reduced into a triangular form thanks to some algorithms.
- One of those is the so-called “Gaussian elimination”.
- It is based on the idea of reducing the matrix to a triangular form through linear combinations of rows or columns.

Gaussian elimination

- Example with a 3x3 matrix:

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

- Starting from the second row, one subtracts the element *ij* of the *i*-th row the quantity: $(a_{1j}/a_{11})a_{i1}$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} - \frac{a_{12}}{a_{11}}a_{21} & a_{23} - \frac{a_{13}}{a_{11}}a_{21} \\ 0 & a_{32} - \frac{a_{12}}{a_{11}}a_{31} & a_{33} - \frac{a_{13}}{a_{11}}a_{31} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 - \frac{b_1}{a_{11}}a_{21} \\ b_3 - \frac{b_1}{a_{11}}a_{31} \end{bmatrix}$$

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \end{bmatrix}$$

Gaussian elimination

- Then we repeat the operation starting from the second row:

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a'_{33} - \frac{a'_{23}}{a'_{22}} a'_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 - \frac{b'_2}{a'_{22}} a'_{23} \end{bmatrix}$$

- That is, we get to a triangular matrix:

$$\begin{bmatrix} a''_{11} & a''_{12} & a''_{13} \\ 0 & a''_{22} & a''_{23} \\ 0 & 0 & a''_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b''_1 \\ b''_2 \\ b''_3 \end{bmatrix}$$

which can be solved with the FS algorithm...

Complexity of GE

- How many operations are needed for a *n x n* matrix?
- To eliminate the first column we need to compute the ratio: a_{1j}/a_{11} for j=2,…,n AND the ratio: b_1/a_{11}, for a total of *n* products…
- Then, for each row i=2,…,n (n-1 rows in total!) we need to multiply these *n* products times a_{i1}!
- Therefore, to cancel the first column we need:

$$n \cdot (n-1) + n = n^2 \text{ products}$$

Complexity of GE

- To eliminate the second column we then need:

  $(n-1)^2$ operations…

- To put the original matrix in a triangular form the needed number of operations is:

$$N = n^2 + (n-1)^2 + \ldots 2^2 + 1 =$$

$$= \sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6} \sim O(n^3)$$

- To finally solve the system, we need to add the

$$n(n+1)/2 \sim O(n^2)$$

  operations needed for the BS on the reduced matrix!

LU factorization

- An alternative method, which however has some advantages over the GE is the so-called “LU-factorization”.
- Given the original system of linear equations:

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

  it is possible to show that, if *A* is invertible, then it is possible to write *A* as the product of two matrices *L* (lower triangular) and *U* (upper triangular), namely:

$$A = LU$$

LU factorization

- How to find the decomposition: *A=LU*?

$$
A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} =
$$

$$
= \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix}
$$

LU factorization

- First, notice that on the LHS we have $n \times n$ numbers, on the RHS we have $n \times n + n$.
- That is, there are n of such values which can be chosen at will! $\longrightarrow$ The decomposition is not unique!!!
- In order to decrease the total number of operations, we can choose two slightly different algorithms:
  - Doolittle's algorithm: $l_{ii}=1$;
  - Crout's algorithm: $u_{ii}=1$.

LU factorization

- When we put this product into the original system, we get:

$$A\mathbf{x} = (LU)\mathbf{x} = L(\underbrace{U\mathbf{x}}_{\mathbf{y}}) = \mathbf{b}$$

and then, if we define: **y**=*U***x**, we finally get the system, equivalent to the original one:

$$U\mathbf{x} = \mathbf{y}$$

$$L\mathbf{y} = \mathbf{b}$$

LU factorization

- Let's see the second (Crout's algorithm, u_{ii}=1)!

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} =$$

$$= \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} 1 & u_{12} & \dots & u_{1n} \\ 0 & 1 & \dots & u_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

LU factorization

- Let’s do the products of *L* and *U*:
- first row: $a_{11} = l_{11};\ a_{12} = l_{11}u_{12};\ \ldots\ ; a_{1n} = l_{11}u_{1n}$

  from which we get:

$$l_{11} = a_{11};\ u_{12} = \frac{a_{12}}{l_{11}};\ \ldots\ ; u_{1n} = \frac{a_{1n}}{l_{11}}$$

- second row

$$\begin{aligned} a_{21} &= l_{21}; \\ a_{22} &= l_{21}u_{12} + l_{22}; \\ a_{23} &= l_{21}u_{13} + l_{22}u_{23}; \\ &\ldots \\ a_{2n} &= l_{21}u_{1n} + l_{22}u_{2n} \end{aligned}$$

LU factorization

from which we can compute l_{21} and u_{2j}:

$$l_{21} = a_{21}; \qquad l_{22} = a_{22} - l_{21}u_{12};$$

$$u_{2j} = \frac{a_{2j} - l_{21}u_{1j}}{l_{22}} \quad \text{for } j = 3, \ldots, n$$

- Third row:

$$\begin{aligned} a_{31} &= l_{31}; \\ a_{32} &= l_{31}u_{12} + l_{32}; \\ a_{33} &= l_{31}u_{13} + l_{32}u_{23} + l_{33}; \\ a_{34} &= l_{31}u_{14} + l_{32}u_{24} + l_{33}u_{34}; \\ &\ldots \\ a_{3n} &= l_{31}u_{1n} + l_{32}u_{2n} + l_{33}u_{3n}. \end{aligned}$$

LU factorization

from which we get the relations:

$$
\begin{aligned}
l_{31} &= a_{31};\\
l_{32} &= a_{32} - l_{31}u_{12};\\
l_{33} &= a_{33} - l_{31}u_{13} - l_{32}u_{23};\\
u_{3j} &= \frac{a_{3j} - l_{31}u_{1j} - l_{32}u_{2j}}{l_{33}} \quad \text{for } i = 4, \ldots, n
\end{aligned}
$$

- By going on writing the relations for the following lines...

LU factorization

… we finally get the final relations for l_{ij} and u_{ij}:

- for $i = 1$: $l_{11} = a_{11}$

$$u_{1j} = \frac{a_{1j}}{l_{11}} \qquad \text{for} \quad j = 2, \ldots, n$$

for $i = 2, \ldots, n$:

$$\left.\begin{aligned} l_{i1} &= a_{i1} \\ l_{ij} &= a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}, \quad j = 2, \ldots, i \\ u_{ij} &= \frac{1}{l_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right), \quad j = i+1, \ldots, n \end{aligned}\right\}$$

Complexity of LU factorization

- How many operations are required to find the coefficients of *L* and *U*?

  the computation of u_{1j} requires *n*-1 products!

- For each *i*:

for the l_{ij}, j=2, …, i	for the u_{ij}, j=i+1, …, n
$$\begin{array}{ccc} 1 & \text{products for} & j=2 \\ 2 & \text{products for} & j=3 \\ \dots & & \dots \\ i-1 & \text{products for} & j=i \end{array}$$	$$\begin{array}{ccc} i & \text{products for} & j=i+1 \\ i+1 & \text{products for} & j=i+2 \\ \dots & & \dots \\ n-1 & \text{products for} & j=n \end{array}$$

Complexity of LU factorization

- Therefore, for each *i* (=2, …,*n*-1) , the computation of l_{ij} and u_{ij} requires:

$$1 + 2 + \ldots + (i-1) + i + (i+1) + \ldots + (n-1) = \frac{(n-1)\cdot n}{2}$$

- this number has to be multiplied for the number of values of *i* and then added up to the number of operations for the first row of u_{ij}, *n*-1, that is:

$$N = (n-1) + (n-1)\cdot\frac{(n-1)\cdot n}{2}$$

$$= (n-1)\cdot\frac{(n^2-n+2)}{2} \sim O(n^3)$$

Advantages of LU factorization

- Hence, the LU factorization has the same complexity as the Gaussian elimination (indeed one could show that the GE is a special case of LU factorization!)...

  ... however ...

  there are cases in which the LU factorization can be **MUCH** more convenient than the GE!

- For instance, a typical case is when one has to solve a set of different linear systems with the same matrix of the coefficients A and different RHSs b_i.

Advantages of LU factorization

- In this case:

$$A\mathbf{x}_i = \mathbf{b}_i, \qquad \text{for } i = 1, \ldots, m$$

- The advantage of the LU factorization over the GE is in the fact that in GE both the matrix A and the vectors of known terms **MUST** be transformed! In LU, ONLY the computation of l_{ij} and u_{ij} is to be carried out the first time, after that only the FS and BS have to be computed to find the solution!

Advantages of LU factorization

- More specifically, for EG we have:

$$m \cdot \left[(n-1) \cdot \frac{(n^2 - n + 2)}{2} + \frac{n(n+1)}{2} \right] \sim O(m \cdot n^3)$$

  operations
- For the LU factorization:

$$(n-1) \cdot \frac{(n^2 - n + 2)}{2} + 2 \cdot m \cdot \frac{n(n+1)}{2} \sim O(n^3 + m \cdot n^2)$$

  operations, which is much better (for instance, when *n*=*m*)!

Advantages of LU factorization

- A typical example is when one is to invert a *nxn* matrix: $A\,A^{-1} = \mathbb{I}$

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} a'_{11} & a'_{12} & \dots & a'_{1n} \\ a'_{21} & a'_{22} & \dots & a'_{2n} \\ \dots & \dots & \dots & \dots \\ a'_{n1} & a'_{n2} & \dots & a'_{nn} \end{bmatrix} = \\ = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

where a'_{ij} are the coefficients of A^{-1}.

Advantages of LU factorization

- One can re-write this as *n* different systems of *nxn* equations in the form:

$$A\mathbf{x}_i = \mathbf{b}_i \qquad \text{for } i = 1, \ldots, n$$

where:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}$$

$$\mathbf{x}_i = \begin{bmatrix} a'_{1i} \\ a'_{2i} \\ \ldots \\ a'_{ni} \end{bmatrix} \qquad \mathbf{b}_i = \begin{bmatrix} \delta_{1i} \\ \delta_{2i} \\ \ldots \\ \delta_{ni} \end{bmatrix}$$

$$\delta_{ij} = \begin{cases} 1 & \text{for } i = j; \\ 0 & \text{for } i \neq j \end{cases}$$ is the **Kronecher** symbol.

Stability of LU factorization

- How does the truncation errors propagate during the resolution of the system?
- One could show that, a **sufficient** condition to avoid instabilities is that the matrix is **diagonally dominant**:

$$|a_{ii}| \geq |a_{ij}| \qquad \forall\ i = 1, \ldots, n;\ j = 1, \ldots, n$$

that is, the coefficients along the diagonal of the matrix *A* must be greater (in absolute value) than the out-of-diagonal coefficients.

Sparse matrices

- We have seen as the methods we have studied till now require ~ $O(n^3)$ operations to solve the system when the coefficients of the system are all different from zero.
- However, many times in numerical analysis, it happens that the matrix *A* of the system to solve has many zeros in determined positions.
- In such cases, we talk of **“sparse matrices”**, in the sense that a non-zero coefficient may appear only in some particular positions of the matrix *A.*

Sparse matrices

- Some examples:

$$A = \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix} \qquad \text{triangular matrix}$$

$$A = \begin{bmatrix} x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x \end{bmatrix} \qquad \text{block matrix}$$

Sparse matrices

$$A = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \end{bmatrix}$$

Hessenberg, quasi-triangular matrix

$$A = \begin{bmatrix} x & x & x & x & 0 & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 \\ x & x & x & x & x & x & 0 & 0 \\ 0 & x & x & x & x & x & x & 0 \\ 0 & 0 & x & x & x & x & x & x \\ 0 & 0 & 0 & x & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & x & x & x \end{bmatrix}$$

band matrix

Sparse matrices

$$A = \begin{bmatrix} x & x & 0 & 0 & x & x & 0 & x \\ x & x & x & 0 & 0 & x & x & 0 \\ x & x & x & x & 0 & 0 & x & x \\ 0 & x & x & x & x & 0 & 0 & x \\ 0 & 0 & x & x & x & x & 0 & 0 \\ 0 & 0 & 0 & x & x & x & x & 0 \\ x & x & 0 & 0 & x & x & x & x \\ x & x & 0 & 0 & 0 & x & x & x \end{bmatrix} \qquad \text{generic sparse matrix}$$

We have already seen the case of triangular matrices, we will see the case of band matrices and, finally, the general case. We will not deal with the other cases (Hessenberg matrices, block matrices, ecc.)

Band matrices

- A case that often appears in numerical analysis is the case in which the matrix of the coefficients of the system has the form of a "**band matrix**" with lower-bandwidth *p* and upper-bandwidth *q*, that is:

$$a_{ij} \neq 0 \quad \text{iff} \quad \begin{cases} p \geq i - j & \text{if } i \geq j; \\ q \geq j - i & \text{if } j \geq i. \end{cases}$$

- The quantity *M*=*p*+*q* is called the **bandwitdh** of the band matrix.

Band matrices

- Example (n=8):

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & 0 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} \\ 0 & 0 & 0 & 0 & a_{75} & a_{76} & a_{77} & a_{78} \\ 0 & 0 & 0 & 0 & 0 & a_{86} & a_{87} & a_{88} \end{bmatrix}$$

- Here: $p = 2$, $q = 3$, $M = p+q = 5$.

Band matrices

- The idea is to solve the system by avoiding the multiplications for 0.
- This is done by using a LU decomposition in which L has only p lower co-diagonals different from zero and U has only q upper co-diagonal different from zero.
- The number of operations needed to solve a band system is:
  - $O[(p+q)n^2]$ for the LU decomposition;
  - $O(pn)+O(qn)$ for the FS and BS.

Tridiagonal matrices

- A specially important case is the one with: *p*=*q*=1, the so-called tridiagonal case.
- In this case, the matrix reads:

$$A = \begin{bmatrix} a_1 & c_1 & 0 & \dots & 0 & 0 \\ b_2 & a_2 & c_2 & \dots & 0 & 0 \\ 0 & b_3 & a_3 & c_3 & \dots & 0 \\ \dots & \dots & \ddots & \ddots & \ddots & \dots \\ 0 & 0 & \dots & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & 0 & 0 & \dots & b_n & a_n \end{bmatrix}$$

Tridiagonal matrices

- which can be factorized (e.g. with the Doolittle algorithm) as:

$$A = \begin{bmatrix} a_1 & c_1 & 0 & \dots & 0 & 0 \\ b_2 & a_2 & c_2 & \dots & 0 & 0 \\ 0 & b_3 & a_3 & c_3 & \dots & 0 \\ \dots & \dots & \ddots & \ddots & \ddots & \dots \\ 0 & 0 & \dots & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & 0 & 0 & \dots & b_n & a_n \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ \beta_2 & 1 & 0 & \dots & 0 \\ \dots & \ddots & \ddots & \dots & \dots \\ 0 & \dots & \beta_{n-1} & 1 & 0 \\ 0 & 0 & \dots & \beta_n & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 & \gamma_1 & 0 & \dots & 0 \\ 0 & \alpha_2 & \gamma_2 & \dots & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \alpha_{n-1} & \gamma_{n-1} \\ 0 & 0 & \dots & 0 & \alpha_n \end{bmatrix}$$

Tridiagonal matrices

- By multiplying the α, β, and γ coefficients as before, one obtains:

$$\alpha_1 = a_1; \qquad \gamma_1 = c_1$$

$$\begin{cases} \beta_i = b_i/\alpha_{i-1} \\ \alpha_i = a_i - \beta_i \gamma_{i-1} \quad \text{for } i = 2, \ldots, n \\ \gamma_i = c_i \end{cases}$$

therefore the solution of the LU system:

$$\begin{cases} L\mathbf{y} & = \mathbf{f} \\ U\mathbf{x} & = \mathbf{y} \end{cases}$$

Tridiagonal matrices

- can be found as:

$$y_1 = f_1$$
$$y_i = f_i - \beta_i y_{i-1}; \quad i = 2, \ldots, n$$

$$x_n = y_n / \alpha_n$$
$$x_i = (y_i - c_i x_{i+1})/\alpha_i; \quad i = n-1, \ldots, 1$$

that is called “**Thomas’ algorithm**” and requires:

$$\underbrace{O[2(n-1)]}_{\textbf{LU factorization}} + \underbrace{O(3n-2)}_{\textbf{FS and BS}}$$

Sparse matrices

- Let us suppose now we have a generic linear system of equations:

$$A\mathbf{x} = \mathbf{b} \qquad (1)$$

  and let us suppose the matrix A is **sparse**, namely it has many elements equal to zero and some elements different from zero in generic, but known, positions *i-j*.

- Our aim is always to find ***x*** that satisfies the relation (1)!

Sparse matrices

- A note on solving generic polynomial equations:
  - When we have an equation in the form:

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-2} x^2 + a_{n-1} x + a_n = 0$$

  - We can rewrite the equation as:

$$a_{n-1} x = -(a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-2} x^2 + a_n) \qquad (2)$$

  - For *n* > 4 we do not know how to solve the equation with algebraic methods, however we can try to find an approximate solution!

Sparse matrices

- The trick is to suppose that we know an approximated value of the solution x_0 which does not satisfy the (2), but we can get an “improved” solution (closer to the real one) by iterating the formula:

$$a_{n-1}x_{k+1} = -(a_0 x_k^n + a_1 x_k^{n-1} + \ldots + a_{n-2}x_k^2 + a_n) \qquad (3)$$

- For instance, let us consider n=2 (second degree equation):

$$ax^2 + bx + c = 0$$

Sparse matrices

- Suppose that we know an approximated value x_0 of the solution, that is:

$$ax_0^2 + bx_0 + c \simeq 0$$

  because x_0 is **NOT** the real solution.

- We can get a better approximation x_1 of the solution as: $x_1 = x_0 + \delta x_0$ where we suppose:

$$\delta x_0 << x_0$$

- By substituting x_1 in the original equation:

$$a(x_0 + \delta x_0)^2 + bx_1 + c = 0$$

$$\Rightarrow \quad bx_1 = -\left[a(x_0 + \delta x_0)^2 + c\right] \sim -\left[ax_0^2 + c\right]$$

Sparse matrices

- Then, we can iterate the procedure, getting:

$$\mathrm{x}_{k+1} = -(ax_k^2 + c)/b$$

- For example, the equation:

$$x^2 + 4x + 3 = 0$$

has solutions: *x*=-1 and *x*=-3.

- If we suppose, for instance, x_0=0, we get the succession of approximated solutions:

$$x_0 = 0; \quad x_1 = -0.75; \quad x_2 = -0.890625; \quad x_3 = -0.948303;$$
$$x_4 = -0.974820; \quad x_5 = -0.987568; \quad x_6 = -0.993823; \ldots$$

Sparse matrices

- Going back to the case of sparse matrices, we can try to use an analogous method to solve a system of linear equations.

- Methods of this kind, in which one searches for a succession of solutions $x^{(k)}$ is called a **Relaxation Method**, in the sense that the solution converges (relaxes) towards the real solution of the system.

Jacobi relaxation method

- Let us suppose that the matrix of coefficients *A* of the original system:

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

can be split in a diagonal part *D* and an off-diagonal part *R*:

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

where:

$$D = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ & & \ddots & \\ \dots & \dots & & \dots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix} \; ; \; R = \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & 0 & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{bmatrix}$$

Jacobi relaxation method

- We have the following relation:

$$D\mathbf{x} = \mathbf{b} - R\mathbf{x}$$

therefore one can write a succession of approximations for the solution *x* in the form:

$$D\mathbf{x}^{(k+1)} = \mathbf{b} - R\mathbf{x}^{(k)}$$

that is:

$$\mathbf{x}^{(k+1)} = D^{-1}[\mathbf{b} - R\mathbf{x}^{(k)}]$$

which always converges to the solution, provided that *D* is invertible (i.e. all a_{ii} are not zero!).

Jacobi relaxation method

- This relation can be written for the generic *i*-th component of the solution vector *x*:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^{(k)} \right]$$

- This formula can be more convenient than the usual LU factorization if:

1) x_0 is close enough to the real solution so that the convergence is reached in few steps;

2) There are only few terms $a_{ij} \neq 0$!

Jacobi relaxation method

- Generally one stops the iteration loop when the difference between two successive approximations of the solution is smaller than a given tolerance *p*, in some norm (e.g.: $|\mathbf{v}| = \sqrt{v_1^2 + \ldots + v_n^2}$)

$$|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}| < p$$

- Example:

$$\begin{bmatrix} 3 & 1 & 0 \\ 0 & 2 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ -2 \end{bmatrix}$$

Jacobi relaxation method

- The exact solution of the system is:

$$x_1 = 1; \quad x_2 = 0; \quad x_3 = -1$$

- Let us suppose that the initial guess for the solution is:

$$\mathbf{x}^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- For the Jacobi's method we obtain the sequence of values:

Jacobi relaxation method

$$\mathrm{x}_1^{(1)} = \frac{1}{a_{11}}\left[b_1 - \sum_{j=1,j\neq i}^{3} a_{1j}x_j^{(0)}\right] = \frac{1}{3}\left\{3 - \left[1\cdot x_2^{(0)}\right]\right\} = 1$$

$$\mathrm{x}_2^{(1)} = \frac{1}{a_{22}}\left[b_2 - \sum_{j=1,j\neq i}^{3} a_{2j}x_j^{(0)}\right] = \frac{1}{2}\left[0 - 0\right] = 0$$

$$\mathrm{x}_3^{(1)} = \frac{1}{a_{33}}\left[b_3 - \sum_{j=1,j\neq i}^{3} a_{3j}x_j^{(0)}\right] = \frac{1}{1}\left\{-2 - \left[-1\cdot x_1^{(0)}\right]\right\} = -2$$

$$\mathrm{x}_1^{(2)} = \frac{1}{a_{11}}\left[b_1 - \sum_{j=1,j\neq i}^{3} a_{1j}x_j^{(1)}\right] = \frac{1}{3}\left\{3 - \left[1\cdot x_2^{(1)}\right]\right\} = 1$$

$$\mathrm{x}_2^{(2)} = \frac{1}{a_{22}}\left[b_2 - \sum_{j=1,j\neq i}^{3} a_{2j}x_j^{(1)}\right] = \frac{1}{2}\left[0 - 0\right] = 0$$

$$\mathrm{x}_3^{(2)} = \frac{1}{a_{33}}\left[b_3 - \sum_{j=1,j\neq i}^{3} a_{3j}x_j^{(1)}\right] = \frac{1}{1}\left\{-2 - \left[-1\cdot x_1^{(1)}\right]\right\} = -1$$

Jacobi relaxation method

- Notice that:
  - We found the **exact** solution after just **2 steps**: this does not happen usually (only in very simple cases like the one we are considering!), because **the solution is usually approximated** and **several steps** are required to get the solution with the necessary precision;
  - We kept into account in the products **only the terms** which are actually **different from zero**, therefore **we need to know their position** on each row of the matrix!

Gauss-Seidel relaxation method

- Another very common method is the **Gauss-Seidel** method which consists in splitting the original A matrix of the system in a **diagonal**, plus a **lower** and a **upper triangular** matrices:

$$A\mathbf{x} = \mathbf{b}, \qquad A = D + L + U$$

$$D = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \ddots & \dots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}; \quad L = \begin{bmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ \dots & \dots & 0 & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & a_{2n} \\ \dots & \dots & 0 & \dots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

Gauss-Seidel relaxation method

- Then we put the *U* term on the RHS:

$$(D+L)\mathbf{x} = (\mathbf{b} - U\mathbf{x})$$

- We know how to solve the LHS of thus system (for instance with the forward substitutions!), then we find the solution as the succession of approximations:

$$(D+L)\mathbf{x}^{(k+1)} = (\mathbf{b} - U\mathbf{x}^{(k)})$$

that is:

$$D\mathbf{x}^{(k+1)} = (\mathbf{b} - L\mathbf{x}^{(k+1)} - U\mathbf{x}^{(k)})$$

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - L\mathbf{x}^{(k+1)} - U\mathbf{x}^{(k)})$$

Gauss-Seidel relaxation method

- Written in terms of the elements of the vector of solutions:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

- For instance, we can apply to the previous system:

$$\begin{bmatrix} 3 & 1 & 0 \\ 0 & 2 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ -2 \end{bmatrix}$$

Gauss-Seidel relaxation method

$$\mathrm{x}_1^{(1)} = \frac{1}{a_{11}}\left[b_1 - 0 - \sum_{j=2}^{3} a_{1j}x_j^{(0)}\right] = \frac{1}{3}\left\{3 - \left[1\cdot x_2^{(0)}\right]\right\} = 1$$

$$\mathrm{x}_2^{(1)} = \frac{1}{a_{22}}\left[b_2 - \sum_{j=1}^{1} a_{2j}x_j^{(1)} - \sum_{j=3}^{3} a_{2j}x_j^{(0)}\right] = \frac{1}{2}\left[0 - 0 - 0\right] = 0$$

$$\mathrm{x}_3^{(1)} = \frac{1}{a_{33}}\left[b_3 - \sum_{j=1}^{2} a_{3j}x_j^{(1)} - 0\right] = \frac{1}{1}\left\{-2 - \left[-1\cdot x_1^{(1)}\right]\right\} = -1$$

- Some notes:
  - Usually a smaller number of iterations is required (just 1 in this simple case!)
  - However, the iterations cannot be performed in parallel, which limits the application of the method to parallel computing!

Appendix

- Here we show that: $\sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6}$

$$
\begin{aligned}
\sigma &= 1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2 = \\
&= \underbrace{n + n + n + \ldots + n + n}_{n\ \text{times}} + \\
&+ \underbrace{(n-1) + (n-1) + (n-1) + \ldots + (n-1)}_{n-1\ \text{times}} + \\
&+ \ldots + \\
&+ \underbrace{3 + 3 + 3}_{3\ \text{times}} + \\
&+ \underbrace{2 + 2}_{2\ \text{times}} + \\
&+ 1
\end{aligned}
$$

Appendix

- By adding up vertically all the terms on the RHS of the last relation:

$$\sigma = \sum_{i=1}^{n} i + \left[\sum_{i=1}^{n} i - \sum_{i=1}^{1} i\right] +$$

$$+ \left[\sum_{i=1}^{n} i - \sum_{i=1}^{2} i\right] + \left[\sum_{i=1}^{n} i - \sum_{i=1}^{3} i\right] +$$

$$+ \ldots +$$

$$+ \left[\sum_{i=1}^{n} i - \sum_{i=1}^{n-2} i\right] + \left[\sum_{i=1}^{n} i - \sum_{i=1}^{n-1} i\right]$$

Appendix

- The first terms in parentheses add up n times, and the second terms can be grouped as:

$$\sigma = n \sum_{i=1}^{n} i - \sum_{j=1}^{n-1} \sum_{i=1}^{j} i$$

- If we remember that $s = \sum_{i=1}^{m} i = \frac{m(m+1)}{2}$

  we find:

$$\sigma = n \frac{n(n+1)}{2} + \sum_{j=1}^{n-1} \frac{j(j+1)}{2} =$$

$$= \frac{n^2(n+1)}{2} - \frac{1}{2} \sum_{j=1}^{n-1} j^2 - \frac{1}{2} \sum_{j=1}^{n-1} j$$

Appendix

- The second term can be written as:

$$\sum_{j=1}^{n-1} j^2 = \sum_{j=1}^{n} j^2 - n^2 = \sigma - n^2$$

therefore we have:

$$\sigma = \frac{n^2(n+1)}{2} - \frac{1}{2}\left(\sigma - n^2\right) - \frac{1}{2}\frac{(n-1)n}{2} =$$

$$= \frac{n^2(n+1)}{2} - \frac{\sigma}{2} + \frac{n^2}{2} - \frac{(n-1)n}{4}$$

Appendix

- Finally, by bringing the term in σ on the RHS to the LHS, we find:

$$\frac{3}{2}\sigma = \frac{n^2(n+1)}{2} + \frac{n^2}{2} - \frac{n^2-n}{4} \Rightarrow$$

$$\sigma = \frac{2}{3}\left[\frac{2n^3+2n^2+2n^2-n^2+n}{4}\right] =$$

$$= \frac{2n^3+3n^2+n}{6} \qquad \text{Q.E.D.}$$