LINEAR SYSTEMS (2)

Intensive Computation 2019-2020

prof. Annalisa Massini

Viviana Arrigoni

EXACT METHODS:

1. GAUSSIAN ELIMINATION.
2. CHOLESKY DECOMPOSITION.

ITERATIVE METHODS:

1. JACOBI.
2. GAUSS-SEIDEL

CHOLESKY DECOMPOSITION

Direct method to solve linear systems, $Ax=b$.

If A is **symmetric** and **positive-definite**, it can be written as the product of a lower triangular matrix L and its transpose, $A=LL^T$.
L is the **Cholesky factor** of A.

More notions on positive-definite matrices:

A matrix is positive-definite iff its eigenvalues are positive
iff its principal minors are positive (**Silvester's criterion**).

Eigenvalues: solutions of the characteristic polynomial (spectrum).

Principal minors: determinants of the sub matrices on the principal diagonal.

BRIEF NOTE ON THE GEOMETRIC MEANING OF POSITIVE-DEFINITE MATRICES

Since we have already met positive-definite matrices, let's try to visualise them in $\mathbb{R}^2$.

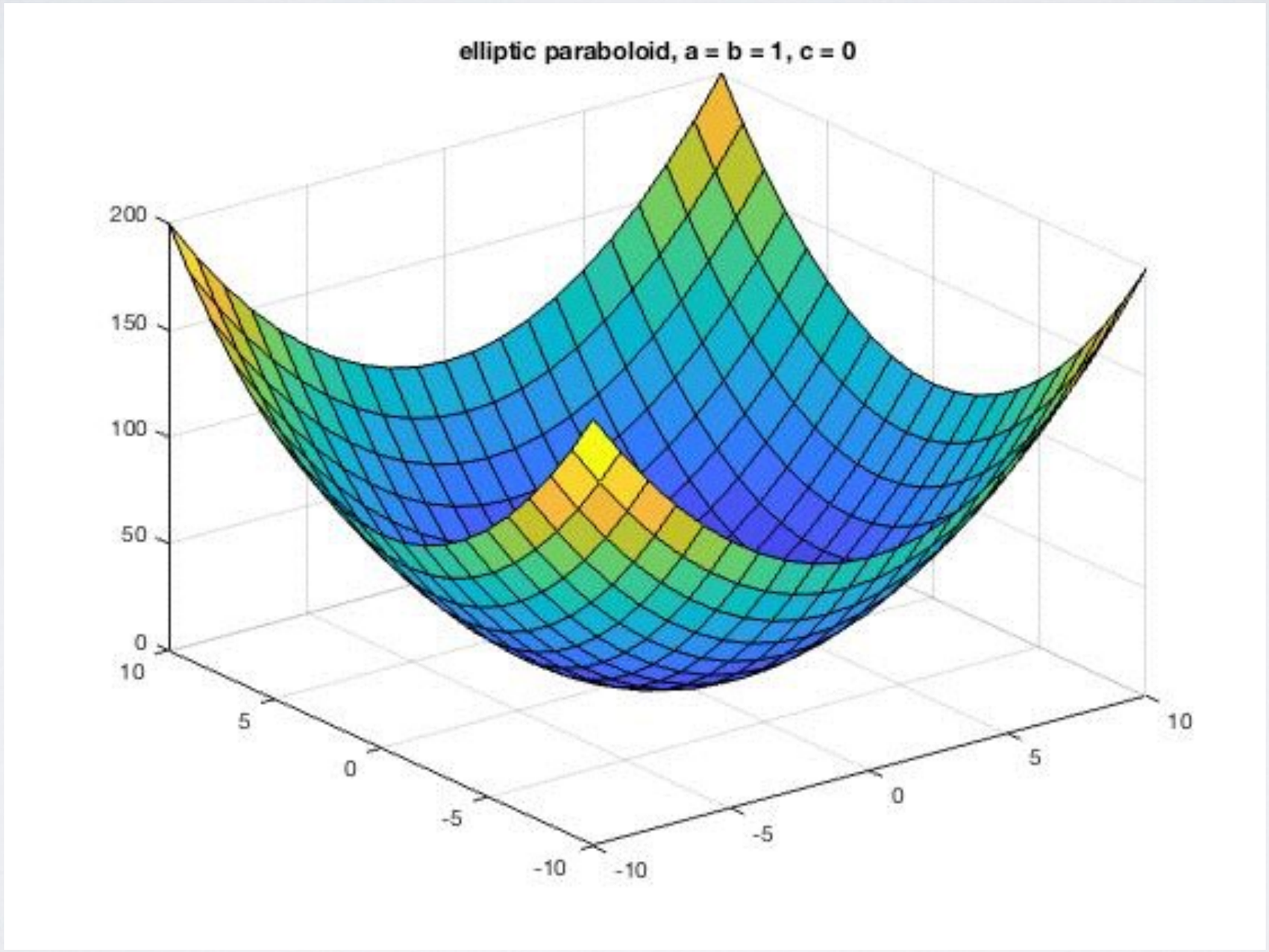
$$\begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} a & c \\ c & b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = ax^2 + 2cxy + by^2$$

→ $> 0,\ \forall (x, y) \neq (0, 0)$
if A is positive-definite

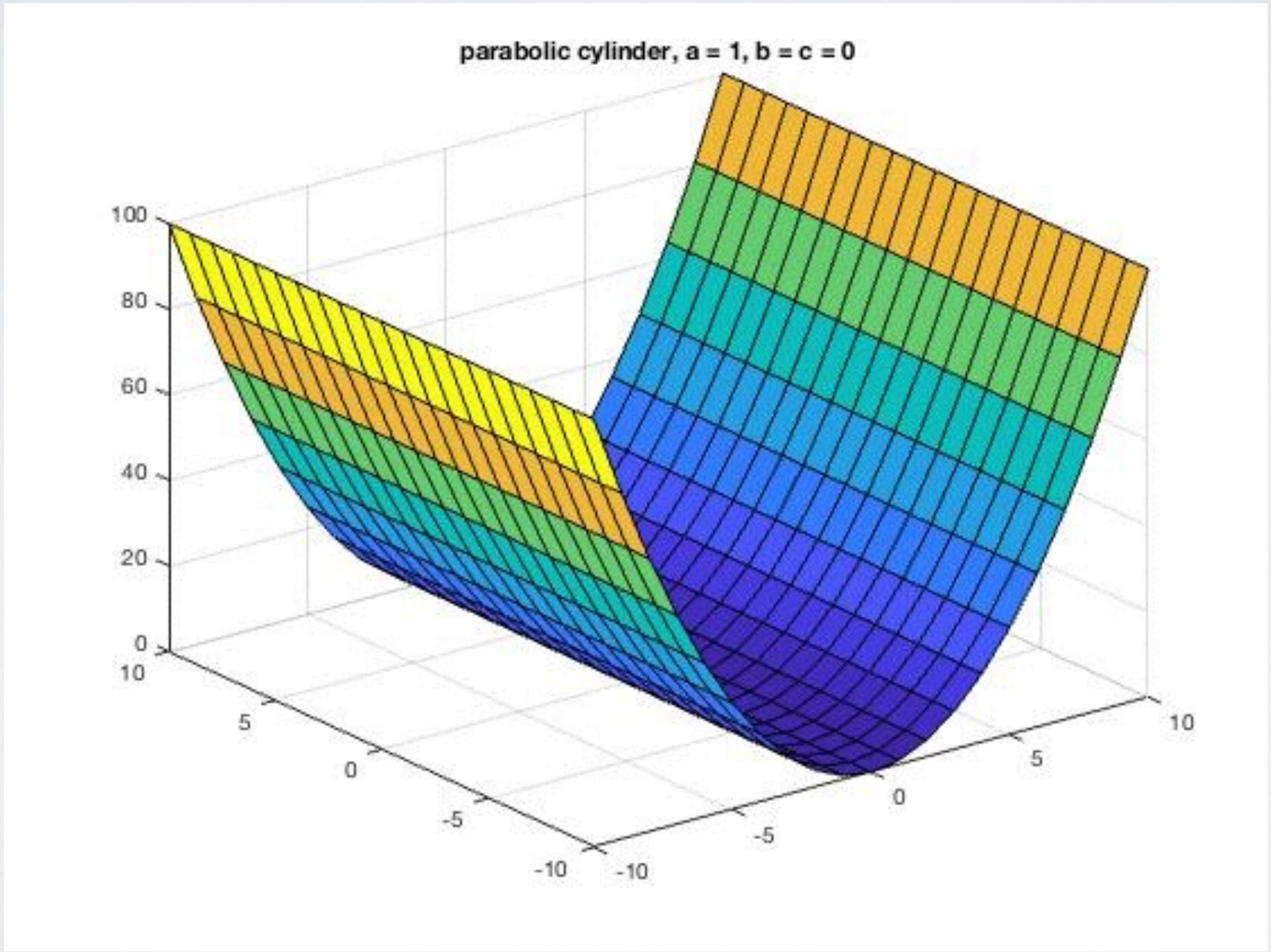
→ $\geq 0,\ \forall (x, y) \neq (0, 0)$
if A is positive-semidefinite

→ $\gneqq 0$
if A is indefinite

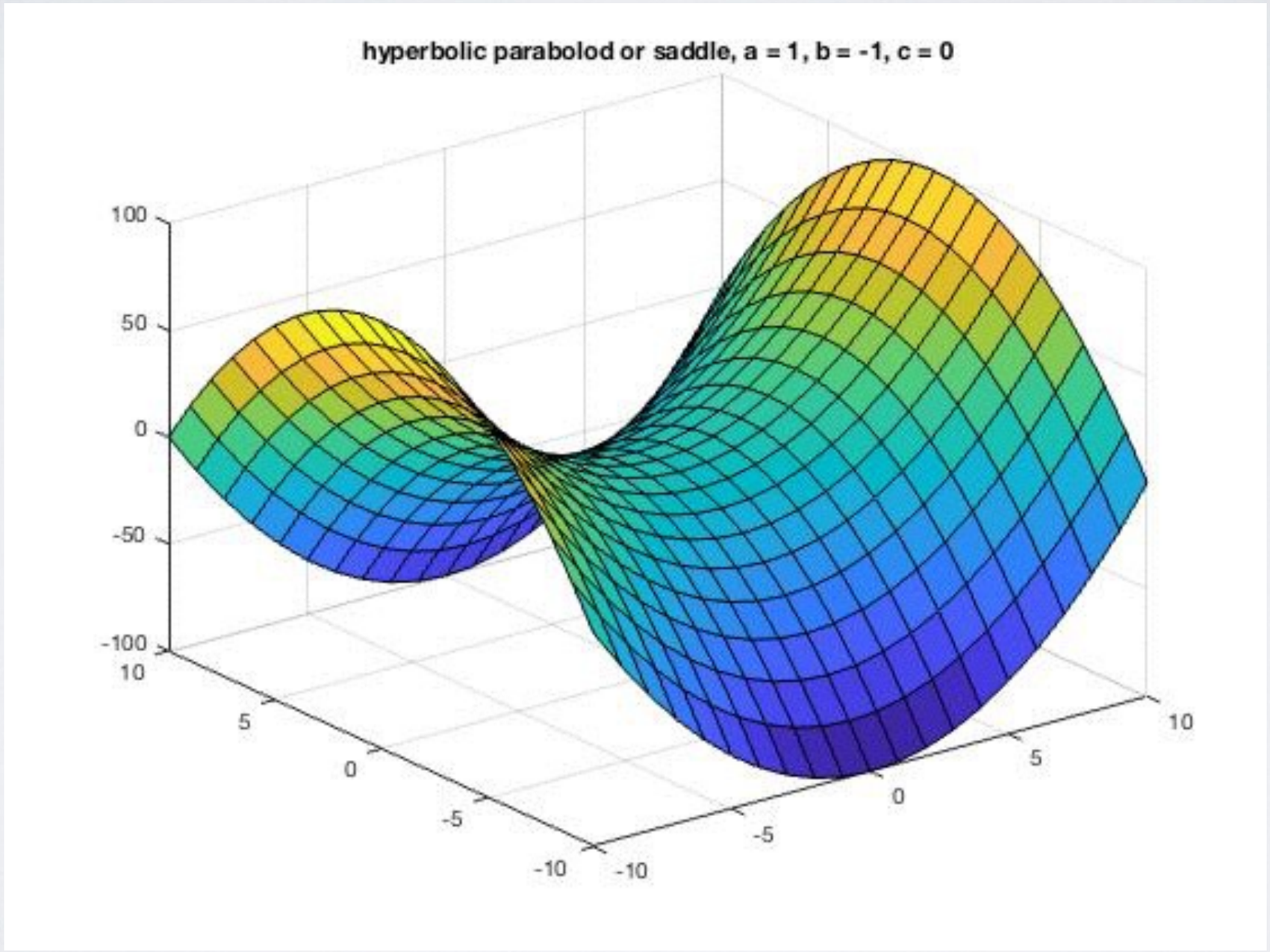
POSITIVE-DEFINITE: ELLIPTIC PARABOLOID



POSITIVE-SEMIDEFINITE: PARABOLIC CYLINDER



INDEFINITE:
HYPERBOLIC PARABOLOID (SADDLE)



CHOLESKY DECOMPOSITION

1° step: Given a symmetric square matrix A, we can write it as follows:

A_{21}^T

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \equiv \begin{pmatrix} a_{1,1} & A_{2,1}^T \\ A_{2,1} & A_{2,2} \end{pmatrix}$$

A_{21} A_{22}

$$A_{2,1} \in \mathbb{R}^{n-1}$$
$$A_{2,2} \in \mathbb{R}^{n-1 \times n-1}$$

Now we want to write A as LL^T where L is lower-triangular and can be written as follows:

$$\begin{pmatrix} l_{1,1} & 0 & \dots & 0 \\ l_{2,1} & l_{2,2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \dots & l_{n,n} \end{pmatrix} \equiv \begin{pmatrix} l_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}$$

where L_{22} is again lower-triangular.

CHOLESKY DECOMPOSITION

So $A=LL^T$ is:

$$\begin{pmatrix} a_{1,1} & A_{2,1}^T \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} l_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix} \begin{pmatrix} l_{1,1} & L_{2,1}^T \\ 0 & L_{2,2}^T \end{pmatrix}$$

$$= \begin{pmatrix} l_{1,1}^2 & l_{1,1}L_{2,1}^T \\ l_{1,1}L_{2,1} & L_{2,1}L_{2,1}^T + L_{2,2}L_{2,2}^T \end{pmatrix}$$

Remember!
We are looking for the entries of L!

At this point we can compute the following portions of L:

$$l_{1,1} = \sqrt{a_{1,1}}$$

$$L_{2,1} = \frac{1}{l_{1,1}} A_{2,1}$$

Basically we get the values of the entries of the first column of L.

Then we compute the following:

$$A_{2,2} = L_{2,1}L_{2,1}^T + L_{2,2}L_{2,2}^T \implies L_{2,2}L_{2,2}^T = A_{2,2} - L_{2,1}L_{2,1}^T =: A^{(2)}$$

CHOLESKY DECOMPOSITION

2° step: repeat step 1 on the matrix $A^{(2)}=L_{2,2}\,L_{2,2}^{T}$.

$$\begin{pmatrix} a_{1,1}^{(2)} & A_{2,1}^{(2)^T} \\ A_{2,1}^{(2)} & A_{2,2}^{(2)} \end{pmatrix} = \begin{pmatrix} l_{2,2} & 0 \\ L_{3,2} & L_{3,3} \end{pmatrix} \begin{pmatrix} l_{2,2} & L_{3,2}^T \\ 0 & L_{3,3}^T \end{pmatrix} = \begin{pmatrix} l_{2,2}^2 & l_{2,2}L_{3,2}^T \\ l_{2,2}L_{3,2} & L_{3,2}L_{3,2}^T + L_{3,3}L_{3,3}^T \end{pmatrix}$$

$$A^{(2)} \in \mathbb{R}^{(n-1)\times(n-1)}$$
$$A_{2,1}^{(2)}, L_{3,2} \in \mathbb{R}^{(n-2)}$$
$$A_{2,2}^{(2)}, L_{3,3} \in \mathbb{R}^{(n-2)\times(n-2)}$$

$$l_{2,2} = \sqrt{a_{1,1}^{(2)}}$$
$$L_{3,2} = \frac{1}{l_{2,2}} A_{2,1}^{(2)}$$

$$A_{2,2}^{(2)} = L_{3,2}L_{3,2}^T + L_{3,3}L_{3,3}^T \implies L_{3,3}L_{3,3}^T = A_{2,2}^{(2)} - L_{3,2}L_{3,2}^T =: A^{(3)}$$

Iterate until the last sub matrix $A^{(n)}$ is considered.
At every step, the size of $A^{(k)}$ decreases of one and the k° column of L is computed.

CHOLESKY DECOMPOSITION

Once we get L, the linear system Ax=b can be written as $LL^Tx=b$, so:
- Solve **Ly=b** using **forward substitution**;
- Solve **$L^Tx=y$** using **backward substitution**.

Why does A have to be positive-definite?

If $A^{(k)}$ is positive-definite, then:

- $a_{1,1}^{(k)} > 0$ (it is its first principal minor), hence $l_{k,k} = \sqrt{a_{1,1}^{(k)}}$ is well defined.

- $A^{(k+1)} = A_{2,2}^{(k)} - \frac{1}{a_{1,1}^{(k)}} A_{2,1} A_{2,1}^T$ is positive-definite.

$$\forall k = 1, \ldots, n$$

AN EXAMPLE...

$$A = \begin{pmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{pmatrix} = \begin{pmatrix} l_{1,1} & 0 & 0 \\ l_{2,1} & l_{2,2} & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} \end{pmatrix} \begin{pmatrix} l_{1,1} & l_{2,1} & l_{3,1} \\ 0 & l_{2,2} & l_{3,2} \\ 0 & 0 & l_{3,3} \end{pmatrix} =$$

$$= \begin{pmatrix} l_{1,1}^2 & l_{1,1}l_{2,1} & l_{1,1}l_{3,1} \\ l_{1,1}l_{2,1} & l_{2,1}^2 + l_{2,2}^2 & l_{2,1}l_{3,1} + l_{2,2}l_{3,2} \\ l_{1,1}l_{3,1} & l_{2,1}l_{3,1} + l_{2,2}l_{3,2} & l_{3,1}^2 + l_{3,2}^2 + l_{3,3}^2 \end{pmatrix}$$

step 1

$$l_{1,1} = \sqrt{25} = 5 \qquad L_{2,1} = \begin{pmatrix} l_{2,1} \\ l_{3,1} \end{pmatrix} = \frac{1}{5} \begin{pmatrix} 15 \\ -5 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \end{pmatrix}$$

Thus $\begin{pmatrix} 5 \\ 3 \\ -1 \end{pmatrix}$ is the first column of L. The matrix for the next iteration is:

$$L_{2,2}L_{2,2}^T = \begin{pmatrix} 18 & 0 \\ 0 & 11 \end{pmatrix} - \begin{pmatrix} 3 \\ -1 \end{pmatrix} \begin{pmatrix} 3 & -1 \end{pmatrix} = \begin{pmatrix} 18 & 0 \\ 0 & 11 \end{pmatrix} - \begin{pmatrix} 9 & -3 \\ -3 & 1 \end{pmatrix} = \begin{pmatrix} 9 & 3 \\ 3 & 10 \end{pmatrix}$$

$A^{(2)}$

…AN EXAMPLE…

step 2

Repeat the same procedure on matrix $A^{(2)}$: $\begin{pmatrix} 9 & 3 \\ 3 & 10 \end{pmatrix}$

$$\begin{pmatrix} 9 & 3 \\ 3 & 10 \end{pmatrix} = \begin{pmatrix} l_{2,2} & 0 \\ l_{3,2} & l_{3,3} \end{pmatrix} \begin{pmatrix} l_{2,2} & l_{3,2} \\ 0 & l_{3,3} \end{pmatrix} = \begin{pmatrix} l_{2,2}^2 & l_{2,2}l_{3,2} \\ l_{2,2}l_{3,2} & l_{3,2}^2 + l_{3,3}^2 \end{pmatrix}$$

Thus:

$$l_{2,2} = \sqrt{9} = 3 \quad L_{3,2} = l_{3,2} = \tfrac{1}{3} \cdot 3 = 1$$

So the second column of L is $\begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix}$, while:

$A^{(3)} := 10 - 1 \cdot 1 = 9$

…AN EXAMPLE…

step 3

$A^{(3)} = 9 = L_{3,3}L_{3,3}^T = l_{3,3}^2$ hence: $l_{3,3} = \sqrt{9} = 3$

So the last column of L is $\begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix}$

In conclusion:

$$A = LL^T = \begin{pmatrix} 5 & 0 & 0 \\ 3 & 3 & 0 \\ -1 & 1 & 3 \end{pmatrix} \begin{pmatrix} 5 & 3 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{pmatrix}$$

…AN EXAMPLE

Assume $b = (1, \ldots, 1)^T$

forward substitution

$$Ly = b \quad \begin{pmatrix} 5 & 0 & 0 \\ 3 & 3 & 0 \\ -1 & 1 & 3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 1.333 \\ 3.556 \end{pmatrix}$$

backward substitution

$$L^T x = b \quad \begin{pmatrix} 5 & 3 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 1.333 \\ 3.556 \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0.0607 \\ 0.0049 \\ 0.1185 \end{pmatrix}$$

CHOLESKY DECOMPOSITION

Algorithm:

Given a linear system Ax=b, A is symmetric and positive-definite and its size is n

- Initialize L as a nxn zero matrix;
- For i=1:n

  L(i,i)=$\sqrt{A(i,i)}$;

  - If i < n

    L(i+1:n,i) = A(i+1:n,i) / L(i,i);
    A(i+i:n,i+1:n)=A(i+1:n,i+1:n) - L(i+1:n,i) * L(i+1:n,i)^T;

- Solve Ly=b (**forward substitution**);
- Solve L^{T}x=y (**backward substitution**);

$O(n^3)$ - expensive!

CHOLESKY FOR SPARSE MATRICES

If A is sparse, so may be L.

If L is sparse, the cost of factorization is less than $n^3/3$ and it depends on n, the number of nonzero elements, and the sparsity pattern. .

100% Fill-in example:

Factorization

$$\begin{pmatrix} 1 & a^T \\ a & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} 1 & a^T \\ a & I \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ a & L_{2,2} \end{pmatrix} \begin{pmatrix} 1 & a^T \\ 0 & L_{2,2}^T \end{pmatrix}$$

$$\begin{pmatrix} * & * & * & * & * & * \\ * & * & & & & \\ * & & * & & & \\ * & & & * & & \\ * & & & & * & \\ * & & & & & * \end{pmatrix} = \begin{pmatrix} * & & & & & \\ * & * & & & & \\ * & * & * & & & \\ * & * & * & * & & \\ * & * & * & * & * & \\ * & * & * & * & * & * \end{pmatrix} \begin{pmatrix} * & * & * & * & * & * \\ & * & * & * & * & * \\ & & * & * & * & * \\ & & & * & * & * \\ & & & & * & * \\ & & & & & * \end{pmatrix} \quad \bullet \bullet \bullet$$

CHOLESKY FOR SPARSE MATRICES

… This can be solved by shifting columns of A of one position to the left (and swapping entries on vectors x and b), getting no fill-in at all.

Shift

$$\begin{pmatrix} 1 & a^T \\ a & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \Rightarrow \begin{pmatrix} I & a \\ a^T & 1 \end{pmatrix} \begin{pmatrix} y \\ x \end{pmatrix} = \begin{pmatrix} c \\ b \end{pmatrix} \Rightarrow$$

Factorization

$$\Rightarrow \begin{pmatrix} I & a \\ a^T & 1 \end{pmatrix} = \begin{pmatrix} I & 0 \\ a^T & \sqrt{1 - a^T a} \end{pmatrix} \begin{pmatrix} I & a \\ 0 & \sqrt{1 - a^t a} \end{pmatrix}$$

$$\begin{pmatrix} * & & & & & * \\ & * & & & & * \\ & & * & & & * \\ & & & * & & * \\ & & & & * & * \\ * & * & * & * & * & * \end{pmatrix} = \begin{pmatrix} * & & & & & \\ & * & & & & \\ & & * & & & \\ & & & * & & \\ & & & & * & \\ * & * & * & * & * & * \end{pmatrix} \begin{pmatrix} * & & & & & * \\ & * & & & & * \\ & & * & & & * \\ & & & * & & * \\ & & & & * & * \\ & & & & & * \end{pmatrix}$$

CHOLESKY FOR SPARSE MATRICES

We have seen that permuting the entries of A can prevent fill-in during factorization, so one can factorize a permutation of A instead of A itself.

Permutation matrix: square matrix having exactly one 1 on every row and column.

If P is a permutation matrix, then PA permutes A's row, AP permutes A's column, P^TAP permutes A's rows and columns.

Permutation matrices are orthogonal: $P^T = P^{-1}$.

Instead of factorizing A, one can factorize P^TAP in order to prevent fill-in

P effects the sparsity pattern of A and it is not known a-priori, but there are heuristic methods to select good permutation matrices.

FACTS ON CHOLESKY DECOMPOSITION

Speaking of methods to avoid pivoting, as the transpose method produces positive-definite matrices, the Cholesky decomposition may be applied.

Gaussian Elimination vs Cholesky Decomposition:

- The Gaussian elimination can be applied to any matrix, even to non square ones, while the Cholesky decomposition can be applied only on (square) symmetric and positive-definite matrices.

- They have the same asymptotic computational cost (O(n3)), but Cholesky is faster by a factor 2.

EXACT METHODS:

1. GAUSSIAN ELIMINATION.
2. CHOLESKY DECOMPOSITION.

ITERATIVE METHODS:

1. JACOBI.
2. GAUSS-SEIDEL

ITERATIVE METHODS

Iterative methods for solving a linear system $Ax=b$ consist in finding a series of **approximate** solutions x^1, x^2, etc., starting from an initial approximate solution x^0 until convergence to the exact solution.

Iteration can be interrupted when the desired **precision** is reached.

For example: precision threshold, $\epsilon = 10^{-3}$.

Suppose the error is computed as the absolute value of the difference of the exact and the approximate solutions element-wise, $|x_i - x^k_i|$.

It follows that an approximate solution x^k is accepted if $|x_i - x^k_i| < \epsilon$.

Assume the solution of $Ax=b$ is $x=(1,\ldots,1)^T$, the approximate solution $x^k=(1.0009,\ldots,1.0009)^T$ satisfies the threshold.

ITERATIVE METHODS

Iterative methods are usually **faster** than exact methods, specially if the coefficient matrix is large and sparse.

One can 'play' with the threshold to find the desired **tradeoff** between speed and accuracy (increasing the precision threshold reduces the number of iterations to reach acceptable approximate solutions, but at the same time it produces less accurate solutions).

On the other hand, for some problems **convergence may be very slow** or the solution may **not converge at all**.

EXACT METHODS:

1. GAUSSIAN ELIMINATION.
2. CHOLESKY DECOMPOSITION.

ITERATIVE METHODS:

1. JACOBI.
2. GAUSS-SEIDEL

JACOBI METHOD

Strictly diagonally dominant matrix:
The absolute value of the diagonal entries is strictly greater than the sum of the absolute value of the non diagonal entries of the corresponding rows.

$$|a_{i,i}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}| \qquad \forall i = 1 \ldots n$$

The Jacobi method always succeeds if A or A^T are strictly diagonally dominant.

* if the system is not strictly diagonally dominant, it may still converge. We will consider strictly diagonally dominant linear systems.

JACOBI METHOD

Given the linear system:

$$
\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n &= b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n &= b_2 \\
&\vdots \\
a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n}x_n &= b_n
\end{aligned}
$$

where A is strictly diagonally-dominant, it can be rewritten as follows by isolating x_i in the i° equation:

$$
\begin{aligned}
x_1 &= \frac{b_1}{a_{1,1}} - \frac{a_{1,2}}{a_{1,1}}x_2 - \ldots - \frac{a_{1,n}}{a_{1,1}}x_n \\
x_2 &= \frac{b_2}{a_{2,2}} - \frac{a_{2,1}}{a_{2,2}}x_1 - \ldots - \frac{a_{2,n}}{a_{2,2}}x_n \\
&\vdots \\
x_n &= \frac{b_n}{a_{n,n}} - \frac{a_{n,1}}{a_{n,n}}x_1 - \frac{a_{n,2}}{a_{n,n}}x_2 - \ldots
\end{aligned}
$$

JACOBI METHOD

Let $x^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)})$ be the initial approximate solution (commonly $x^{(0)}$ is the zero vector).
Find the approximate solution $x^{(1)}$ by substituting $x^{(0)}$ in the right hand side of the linear system:

$$
\begin{aligned}
x_1^{(1)} &= \frac{b_1}{a_{1,1}} - \frac{a_{1,2}}{a_{1,1}} x_2^{(0)} - \ldots - \frac{a_{1,n}}{a_{1,1}} x_n^{(0)} \\
x_2^{(1)} &= \frac{b_2}{a_{2,2}} - \frac{a_{2,1}}{a_{2,2}} x_1^{(0)} - \ldots - \frac{a_{2,n}}{a_{2,2}} x_n^{(0)} \\
&\vdots \\
x_n^{(1)} &= \frac{b_n}{a_{n,n}} - \frac{a_{n,1}}{a_{n,n}} x_1^{(0)} - \frac{a_{n,2}}{a_{n,n}} x_2^{(0)} - \ldots
\end{aligned}
$$

Then use the approximate solution $x^{(1)}$ just computed to find $x^{(2)}$ by substituting $x^{(1)}$ in the right hand side of the linear system.

JACOBI METHOD

Iterate.
At the k° step one finds the approximate solution $x^{(k)}$ by substituting the previous one in the right hand side of the linear system:

$$
\begin{aligned}
x_1^{(k)} &= \frac{b_1}{a_{1,1}} - \frac{a_{1,2}}{a_{1,1}} x_2^{(k-1)} - \dots - \frac{a_{1,n}}{a_{1,1}} x_n^{(k-1)} \\
x_2^{(k)} &= \frac{b_2}{a_{2,2}} - \frac{a_{2,1}}{a_{2,2}} x_1^{(k-1)} - \dots - \frac{a_{2,n}}{a_{2,2}} x_n^{(k-1)} \\
&\vdots \\
x_n^{(k)} &= \frac{b_n}{a_{n,n}} - \frac{a_{n,1}}{a_{n,n}} x_1^{(k-1)} - \frac{a_{n,2}}{a_{n,n}} x_2^{(k-1)} - \dots
\end{aligned}
$$

If A is strictly diagonally-dominant, the produced approximate solutions are more and more accurate.

JACOBI - STOPPING CRITERIA

When should one stop iterating? When the **error** produced is small enough.

Different ways to compute the error at each step:

- $e^{(k)} := ||x^{(k+1)} - x^{(k)}||$: the error is the difference between the last and the previous solutions.
- $e^{(k)} := ||x - x^{(k)}||$: the error is the difference between the exact solution and the approximate solution(exact solution is usually unknown tho).
- $e^{(k)} := ||x^{(k+1)} - x^{(k)}|| / ||x^{(k+1)}||$: the error is the change rate between the last and the previous solutions.

where $||.||$ is the **l^2 norm**: $||x||_2 = \sqrt{(x_1^2 + \ldots + x_n^2)}$.

So the iterations are repeated **while $e^{(k)} >= \epsilon$**

AN EXAMPLE…

The linear system $\begin{aligned} 9x + y + z &= 10 \\ 2x + 10y + 3z &= 19 \\ 3x + 4y + 11z &= 0 \end{aligned}$ has solution $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$

Since A is strictly diagonally dominant, the system can be solved with Jacobi. Compute the error as $e^{(k)} := ||x^{(k+1)} - x^{(k)}||$ and let $x^{(0)} = (0,0,0)$ be the initial approximate solution.

$$\begin{aligned} x^{(1)} &= \tfrac{1}{9}(10 - y^{(0)} - z^{(0)}) = \tfrac{10}{9} \\ y^{(1)} &= \tfrac{1}{10}(19 - 2x^{(0)} - 3z^{(0)}) = \tfrac{19}{10} \\ z^{(1)} &= \tfrac{1}{11}(-3x^{(0)} - 4y^{(0)}) = 0 \end{aligned}$$

So the first approximate solution is $x^{(1)}$: $(x, y, z)^T = (\tfrac{10}{9}, \tfrac{19}{10}, 0)^T$

The error is: $e^{(0)} = 2.2010$.

…AN EXAMPLE…

Substituting $x^{(1)}$ we get:

$$x^{(2)} = \frac{1}{9}(10 - y^{(1)} - z^{(1)}) = \frac{1}{9}(10 - \frac{19}{10}) = \frac{9}{10}$$
$$y^{(2)} = \frac{1}{10}(19 - 2x^{(1)} - 3z^{(1)}) = \frac{1}{10}(19 - 2\frac{10}{9}) = \frac{151}{90}$$
$$z^{(2)} = \frac{1}{11}(-3x^{(1)} - 4y^{(1)}) = \frac{1}{11}(-3\frac{10}{9} - 4\frac{19}{10}) = -\frac{984}{990}$$

and the error is: $e^{(1)}$= 1.0401. One more iteration:

$$x^{(3)} = \frac{1}{9}(10 - y^{(2)} - z^{(2)}) = \frac{1}{9}(10 - \frac{151}{90} + \frac{984}{990}) = 1.03513$$
$$y^{(3)} = \frac{1}{10}(19 - 2x^{(2)} - 3z^{(2)}) = \frac{1}{10}(19 - 2\frac{9}{10} + 3\frac{984}{990}) = \frac{2008}{990}$$
$$z^{(3)} = \frac{1}{11}(-3x^{(2)} - 4y^{(2)}) = \frac{1}{11}(-3\frac{9}{10} - 4\frac{151}{90}) = -\frac{8}{9}$$

The error is: $e^{(2)}$=0.3915

…AN EXAMPLE

Assume the required precision is $\epsilon = 10^{-3}$. Then after 11 iteration the error is 5.9847×10^{-4}. and the approximate solution is

$$\begin{pmatrix} x^{(11)} \\ y^{(11)} \\ z^{(11)} \end{pmatrix} = \begin{pmatrix} 1.0001 \\ 2.0001 \\ -0.9999 \end{pmatrix}$$

JACOBI METHOD

Algorithm:
Given Ax=b, where A is strictly diagonally dominant:

- Initialize the first approximate solution x0 and an array x and set err=Inf and ϵ;
- while err >= ϵ
  - for i = 1:n
    x(i) = b(i);
    - for j = 1:n
      - if j != i
        x(i) = x(i) - A(i,j) * x0(j);
    x(i) = x(i) / A(i,i);
  update err;
  x0 = x;

$O(\#\text{of iterations} \times n^2)$

JACOBI METHOD

If k is the number of iterations (in the while), then the number of operations executed by the Jacobi algorithm is $\mathbf{kn^2}$.
The Gaussian elimination requires $\mathbf{n^3/3}$ operations.

So it is more convenient to use the Jacobi method instead of the Gaussian elimination if $\mathbf{kn^2 < n^3/3}$, that is when $\mathbf{k<n/3}$.
For this reason it is important to assess the minimum number of iterations before deciding whether to apply Jacobi. Such number is the smallest k that satisfies:

$$k > \frac{\log(\frac{\epsilon(1-\lambda)}{\delta_0})}{\log(\lambda)}$$

where

$$\lambda = \max_{i=1,\ldots,n} \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{|a_{i,j}|}{|a_{i,i}|}$$

$$\delta_0 = \max_{i=1,\ldots,n} |x_i^{(0)} - x_i^{(1)}|$$

JACOBI - MATRICIAL FORM

Given Ax=b, where A is strictly diagonally dominant, one can write A as follows:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & & & \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} = \begin{pmatrix} a_{1,1} & 0 & \dots & 0 \\ 0 & a_{2,2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{n,n} \end{pmatrix} - \left[\begin{pmatrix} 0 & 0 & \dots & 0 \\ a_{2,1} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \dots & 0 \end{pmatrix} - \begin{pmatrix} 0 & a_{1,2} & \dots & a_{1,n} \\ 0 & 0 & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \right]$$

$$A \quad = \quad D \quad -[\quad - \quad L \quad - \quad U \quad]$$

So the linear system can be rewritten as: $\{D - [-L - U]\}x = b$

If A is diagonally dominant, D is invertible, so:

$$Dx - [-L - U]x = b \leftrightarrow x - D^{-1}[-L - U]x = D^{-1}b$$
$$\leftrightarrow x = D^{-1}[-L - U]x + D^{-1}b$$

Matricial form of Jacobi:

$$x^{(k+1)} = D^{-1}[-L - U]x^{(k)} + D^{-1}b \qquad k = 0, 1, 2, \dots$$

EXACT METHODS:

1. GAUSSIAN ELIMINATION.
2. CHOLESKY DECOMPOSITION.

ITERATIVE METHODS:

1. JACOBI.
2. GAUSS-SEIDEL

GAUSS-SEIDEL

Iterative method, very similar to Jacobi's.

It always converges if A is strictly diagonally dominant or symmetric and positive-definite.

$$x_1^{(k)} = \frac{b_1}{a_{1,1}} - \frac{a_{1,2}}{a_{1,1}}x_2^{(k-1)} - \frac{a_{1,3}}{a_{1,1}}x_3^{(k-1)} - \ldots - \frac{a_{1,n}}{a_{1,1}}x_n^{(k-1)}$$

$$x_2^{(k)} = \frac{b_2}{a_{2,2}} - \frac{a_{2,1}}{a_{2,2}}x_1^{(k)} - \frac{a_{2,3}}{a_{2,2}}x_3^{(k-1)} - \ldots - \frac{a_{2,n}}{a_{2,2}}x_n^{(k-1)}$$

$$x_3^{(k)} = \frac{b_3}{a_{3,3}} - \frac{a_{3,1}}{a_{3,3}}x_1^{(k)} - \frac{a_{3,2}}{a_{3,3}}x_2^{(k)} - \ldots - \frac{a_{3,n}}{a_{3,3}}x_n^{(k-1)}$$

$$\vdots$$

$$x_n^{(k)} = \frac{b_n}{a_{n,n}} - \frac{a_{n,1}}{a_{n,n}}x_1^{(k)} - \frac{a_{n,2}}{a_{n,n}}x_2^{(k)} - \ldots - \frac{a_{n,n-1}}{a_{n,n}}x_{n-1}^{(k)}$$

* if the system is not strictly diagonally dominant nor positive-definite, it may still converge. We will consider strictly diagonally dominant or positive-definite linear systems.

AN EXAMPLE…

The same linear system we used before:

$$\begin{aligned} 9x + y + z &= 10 \\ 2x + 10y + 3z &= 19 \\ 3x + 4y + 11z &= 0 \end{aligned}$$

Compute the error as $e^{(k)} := ||x^{(k+1)} - x^{(k)}||$ and let $x^{(0)} = (0,0,0)$ be the initial approximate solution.

$$x^{(1)} = \frac{10}{9} - y^{(0)} - z^{(0)} = \frac{10}{9}$$

$$y^{(1)} = \frac{19}{10} - \frac{2}{10}x^{(1)} - \frac{3}{10}z^{(0)} = \frac{19}{10} - \frac{2}{10}\frac{10}{9} = \frac{151}{90}$$

$$z^{(1)} = -\frac{3}{11}x^{(1)} - \frac{4}{11}y^{(1)} = -\frac{3}{11}\frac{10}{9} - \frac{4}{11}\frac{151}{90} = -\frac{904}{990}$$

error = 2.2098.

…AN EXAMPLE…

Second iteration:

$$x^{(2)} = \frac{10}{9} - \frac{1}{9}y^{(1)} - \frac{1}{9}z^{(1)} = \frac{10}{9} - \frac{1}{9}\frac{151}{90} + \frac{1}{9}\frac{904}{990} = \frac{9143}{8910}$$

$$y^{(2)} = \frac{19}{10} - \frac{2}{10}x^{(2)} - \frac{3}{10}z^{(1)} = \frac{19}{10} - \frac{2}{10}\frac{9143}{8910} + \frac{3}{10}\frac{904}{990} = \frac{43853}{22275}$$

$$z^{(2)} = -\frac{3}{11}x^{(2)} - \frac{4}{11}y^{(2)} = -\frac{3}{11}\frac{9143}{8910} - \frac{4}{11}\frac{43853}{22275} = \frac{-487969}{490050}$$

error = 0.3141.

…AN EXAMPLE

If the error threshold is 10^{-3}, after 5 iterations we get:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$$

Computed error: $2.2362 * 10^{-4}$

In iteration 4, the approximate solution was:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1.0002 \\ 2 \\ -1.0001 \end{pmatrix}$$

Computed error: 0.0034

GAUSS-SEIDEL-MATRICIAL FORM

Write A as follows:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & & & \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} = \begin{pmatrix} a_{1,1} & 0 & \dots & 0 \\ 0 & a_{2,2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{n,n} \end{pmatrix} - \left[\begin{pmatrix} 0 & 0 & \dots & 0 \\ a_{2,1} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \dots & 0 \end{pmatrix} - \begin{pmatrix} 0 & a_{1,2} & \dots & a_{1,n} \\ 0 & 0 & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \right]$$

$$A \quad = \quad D \quad -[\quad - \quad L \quad - \quad U \quad]$$

Then the linear system can be written as:

$$(D + L + U)x = b$$

Then: $(D + L)x + Ux = b$

So the matricial form of Gauss-Seidel is:

$$(D + L)x^{(k+1)} = b - Ux^{(k)}$$
$$\forall k = 0, 1, 2, \dots$$

GAUSS-SEIDEL

Algorithm:

Given Ax=b, where A is strictly diagonally dominant or positive-definite:

- Initialize x0, x arrays of length n. Set err=Inf and ϵ;
- while err $\geq \epsilon$
  - for i = 1:n
    x(i) = b(i);
    - if i == 1
      - for j = 2:n
        x(i) = x(i) - A(i,j) * x0(j);
      - else for j = 1:i - 1
        x(i)=x(i) - A(i,j) * x(j);
        - for j = i + 1:n
          x(i) = x(i) - A(i,j) * x0(j);
          …
    …
    x(i) = x(i) \ A(i,i);
    Update err;
    x0 = x;

O(#of iterations x n^2)

JACOBI VS. GAUSS-SEIDEL

On several classes of matrices, Gauss-Seidel converges twice as fast as Jacobi (meaning that at every iteration, the number of fixed exact solution digits that Gauss-Seidel computes is twice as large as Jacobi's).

As soon as the improved entries of the approximate solution are computed, they are immediately used in the same iteration step, k. On the other hand, Jacobi can be parallelized, while Gauss-Seidel is inherently non parallelizable.

The same stopping criteria work for both methods.