LINEAR SYSTEMS (1)

Intensive Computation 2018-19

prof. Annalisa Massini   Viviana Arrigoni
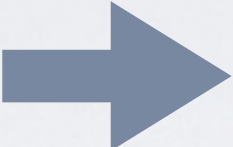
DEFINITIONS (I)

- **Linear system**:

$$a_{1,1}x_1 + \ldots + a_{1,n}x_n = b_1$$
$$\vdots$$
$$a_{m,1}x_1 + \ldots + a_{m,n}x_n = b_m$$

- **Augmented matrix**:

$$[A|b] = \begin{pmatrix} a_{1,1} & \ldots & a_{1,n} & b_1 \\ & \vdots & & \vdots \\ a_{m,1} & \ldots & a_{m,n} & b_m \end{pmatrix}$$

Matricial form:

$$Ax = b$$

$$\begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ & \vdots & \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

- **Square matrix**: # of rows = # of columns.

- **Singular matrix**: square matrix with determinant = 0.

- A square matrix is **invertible** if there exists a matrix such that their product is the identity matrix: $AA^{-1} = A^{-1}A = I$

DEFINITIONS (II)

- **Rank** of a matrix: maximum number of linearly independent rows = maximum number of linearly independent columns.

Example: $rk \begin{pmatrix} 1 & 2 & -2 \\ -2 & -1 & 0 \\ -3 & 3 & 6 \end{pmatrix}$?

DEFINITIONS (II)

- **Rank** of a matrix: maximum number of linearly independent rows = maximum number of linearly independent columns.

Example: $rk \begin{pmatrix} 1 & 2 & -2 \\ -2 & -1 & 0 \\ -3 & 3 & 6 \end{pmatrix} = 2$

The third row can be written as the sum of the first two rows multiplied by 3: $(-3, 3 - 6) - 3[(1, 2, -2) + (-2, -1, 0)] = (0, 0, 0)$

DEFINITIONS (II)

- **Rank** of a matrix: maximum number of linearly independent rows = maximum number of linearly independent columns.

Example: $rk \begin{pmatrix} 1 & 2 & -2 \\ -2 & -1 & 0 \\ -3 & 3 & 6 \end{pmatrix} = 2$

The third row can be written as the sum of the first two rows multiplied by 3: $(-3, 3 - 6) - 3[(1, 2, -2) + (-2, -1, 0)] = (0, 0, 0)$

rk(A)<=min{#rows, #columns}

- If rk(A) = min{#rows, #columns}, A has **full rank**.

A square matrix is nonsingular iff it has full rank

A square matrix is invertible iff it is nonsingular

3

SOLUTIONS

A linear system has one and only one solution iff its coefficient matrix is square and nonsingular

We will consider square and nonsingular matrices

Two linear systems are **equivalent** if they have the same solution.

Applying the following operations to the augmented matrix (A|b) doesn't change the solution of the associated linear system, Ax=b:

- **OP1:** Swap two rows.

- **OP2:** Multiply a row by a nonzero scalar.

- **OP3:** Add to one row a scalar multiple of another.

SOLVING SOME SYSTEMS IS STRAIGHTFORWARD...

- **Diagonal matrix**: $a_{i,j} \neq 0 \implies i = j$

$$\begin{pmatrix} a_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

The solution is:

$$x_i = \frac{b_i}{a_{i,i}} \quad \forall i = 1, \ldots, n$$

- **Upper-triangular matrix**: $a_{i,j} \neq 0 \implies i < j$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

The solution is:

$$x_n = \frac{b_n}{a_{n,n}}$$

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{k=i+1}^{n} a_{i,k} x_k \right)$$

$$i = (n-1, \ldots, 1)$$

SOLVING SOME SYSTEMS IS STRAIGHTFORWARD...

- **Diagonal matrix**: $\qquad a_{i,j} \neq 0 \implies i = j$

$$\begin{pmatrix} a_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

The solution is:

$$x_i = \frac{b_i}{a_{i,i}} \qquad \forall i = 1, \ldots, n$$

- **Upper-triangular matrix**: $\qquad a_{i,j} \neq 0 \implies i < j$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

The solution is:

$$x_n = \frac{b_n}{a_{n,n}}$$
$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{k=i+1}^{n} a_{i,k} x_k \right)$$
$$i = (n-1, \ldots, 1)$$

Backward substitution

SOLVING SOME SYSTEMS IS STRAIGHTFORWARD…

- **Lower-triangular matrix**: $\quad a_{i,j} \neq 0 \implies i > j$

$$\begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{nun} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

The solution is:

$$x_1 = \frac{b_1}{a_{1,1}}$$

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{k=1}^{i-1} a_{i,k} x_k \right)$$

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

- **Orthogonal matrix**: $\quad A^T = A^{-1}$

The solution is: $\quad Ax = b \rightarrow A^T A x = A^T b \rightarrow x = A^T b$

SOLVING SOME SYSTEMS IS STRAIGHTFORWARD…

- **Lower-triangular matrix**: $\quad a_{i,j} \neq 0 \implies i > j$

$$\begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{nun} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

The solution is:

$$x_1 = \frac{b_1}{a_{1,1}}$$

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{k=1}^{i-1} a_{i,k} x_k \right)$$

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

Forward substitution

- **Orthogonal matrix**: $\quad A^T = A^{-1}$

The solution is: $\quad Ax = b \to A^T A x = A^T b \to x = A^T b$

TWO CLASSES OF METHODS TO SOLVE LINEAR SYSTEMS

Direct methods

Find the exact solution (ignoring roundoff) in a finite number of steps.

Gaussian elimination

Cholesky decomposition

Iterative methods

Producing a sequence of approximate solution possibly converging to the exact solutions.

Jacobi method

Gauss-Seidel method

EXACT METHODS:
 1. GAUSSIAN ELIMINATION.
 2. CHOLESKY DECOMPOSITION.
ITERATIVE METHODS:
 1. JACOBI.
 2. GAUSS-SEIDEL

GAUSSIAN ELIMINATION

Idea: Applying repeatedly OP3 to transform the input linear system into an upper-triangular one, then use backward substitution to find the solution.

1° step:

$$
\begin{pmatrix} a_{1,1} \\ a_{2,1} \\ \vdots \\ a_{n,1} \end{pmatrix} \rightarrow \begin{pmatrix} a_{1,1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}
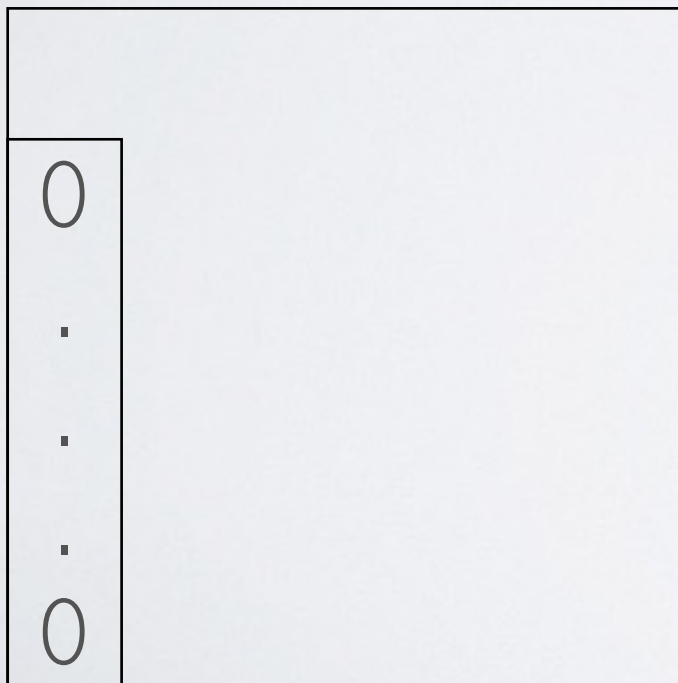$$

k° step:

$$
\begin{pmatrix} a_{1,k} \\ \vdots \\ a_{k,k} \\ a_{k+1,k} \\ \vdots \\ a_{n,k} \end{pmatrix} \rightarrow \begin{pmatrix} a_{1,k} \\ \vdots \\ a_{k,k} \\ 0 \\ \vdots \\ 0 \end{pmatrix}
$$

At every step k, the elements of the k° column with row index > k need to become 0.
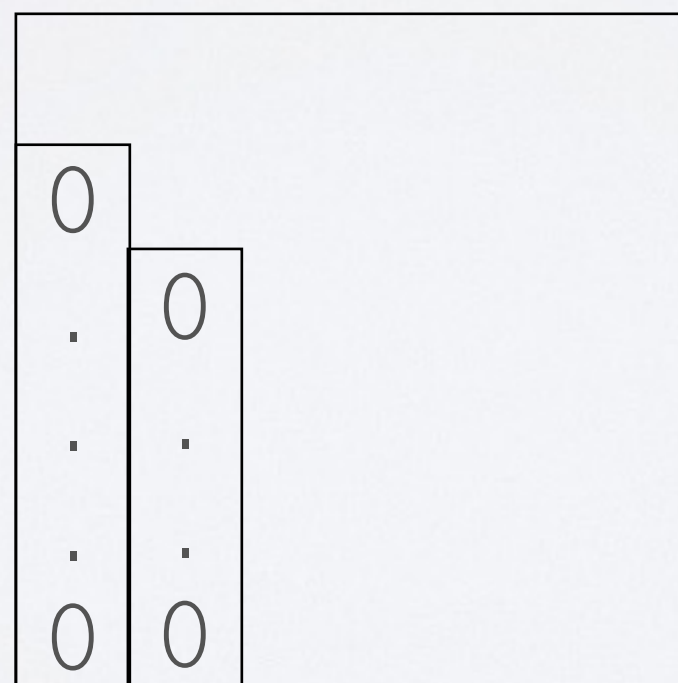
GAUSSIAN ELIMINATION

> Idea: Applying repeatedly OP3 to transform the input linear system into an upper-triangular one, then use backward substitution to find the solution.
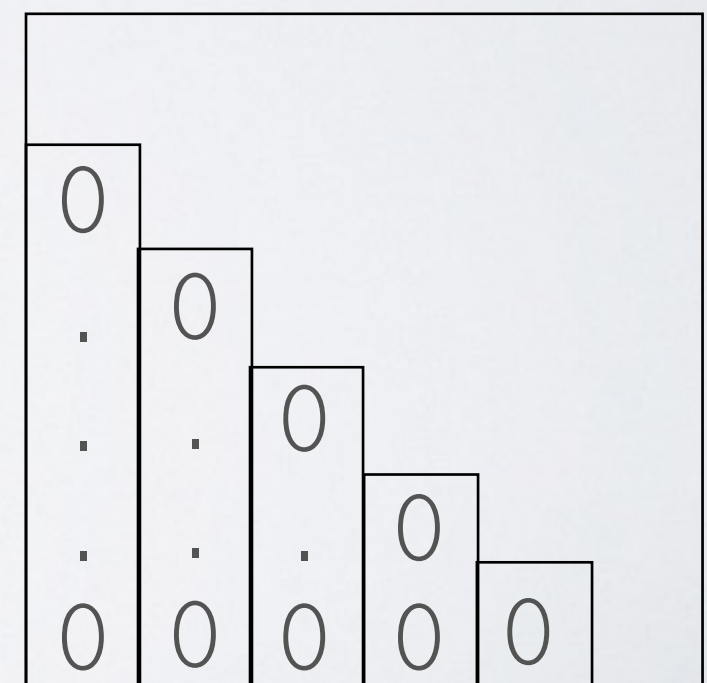
1° step   2° step   (n-1)° step

GAUSSIAN ELIMINATION

Algorithm:

- For every step k: 1 to n-1
  - For every i: k+1 to n (entries of column k with row index > k)

    Compute the **multipliers**: $m_{i,k} = \dfrac{a_{i,k}}{a_{k,k}}$

    - For every j: k to n (entries of the i° rows)

    Subtract the k° row multiplied by $m_{i,k}$ to the i° row:

    $$a_{i,j} \leftarrow a_{i,j} - m_{i,k} a_{k,j}$$

    $$b_i \leftarrow b_i - m_{i,k} b_k$$
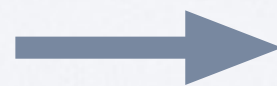
GAUSSIAN ELIMINATION

Algorithm:

- For every step k: 1 to n-1

  - For every i: k+1 to n (entries of column k with row index > k)

    Compute the **multipliers**: $m_{i,k} = \frac{a_{i,k}}{a_{k,k}}$

    - For every j: k to n (entries of the i° rows)

      Subtract the k° row multiplied by m$_{i,k}$ to the i° row:

      $$a_{i,j} \leftarrow a_{i,j} - m_{i,k} a_{k,j}$$ ➡ This will ensure the entries of the k° column with row index i > k to be 0:

      $$b_i \leftarrow b_i - m_{i,k} b_k$$

      $$a_{i,k} - \frac{a_{i,k}}{a_{k,k}} a_{k,k} = 0$$

GAUSSIAN ELIMINATION

Algorithm:

- For every step k: 1 to n-1

  - For every i: k+1 to n (entries of column k with row index > k)

    Compute the **multipliers**: $\quad m_{i,k} = \dfrac{a_{i,k}}{a_{k,k}}$

    - For every j: k to n (entries of the i° rows)

      Subtract the k° row multiplied by $m_{i,k}$ to the i° row:

      $$a_{i,j} \leftarrow a_{i,j} - m_{i,k}a_{k,j}$$

      $$b_i \leftarrow b_i - m_{i,k}b_k$$

- Solve the resulting linear system (backward substitution).

GAUSSIAN ELIMINATION

Algorithm:

$O(n^3)$ - expensive

- For every step k: 1 to n-1

  - For every i: k+1 to n (entries of column k with row index > k)

    Compute the **multipliers**: $m_{i,k} = \dfrac{a_{i,k}}{a_{k,k}}$

    - For every j: k to n (entries of the i° rows)

      Subtract the k° row multiplied by $m_{i,k}$ to the i° row:

      $a_{i,j} \leftarrow a_{i,j} - m_{i,k} a_{k,j}$

      $b_i \leftarrow b_i - m_{i,k} b_k$

- Solve the resulting linear system (backward substitution).

AN EXAMPLE…

$$\begin{pmatrix} 1 & -3 & 1 \\ 2 & -8 & 8 \\ -6 & 3 & -15 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -2 \\ 9 \end{pmatrix}$$

Gaussian Elimination
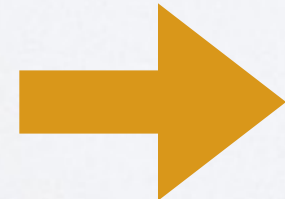
- k°=1° step
  - i = 2: $m_{2,1} = \frac{2}{1} = 2$

$$(2, -8, 8| - 2) - 2(1, -3, 1|4) = (0, -2, 6| - 10)$$

  - i = 3: $m_{3,1} = \frac{-6}{1} = -6$

$$(-6, 3, -15|9) + 6(1, -3, 1|4) = (0, -15, -9|33)$$

- j=1,2,3

$$\begin{pmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & -15 & -9 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ 33 \end{pmatrix}$$

…AN EXAMPLE…

$$\begin{pmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & -15 & -9 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ 33 \end{pmatrix}$$
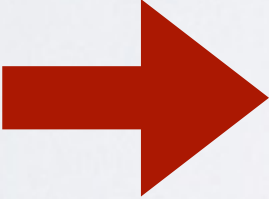
Gaussian Elimination

- k°=2° step

  - i = 3: $m_{3,2} = \frac{-15}{-2} = \frac{15}{2}$

$j = 2,3$

$$(-15, -9|33) - \frac{15}{2}(-2, 6| - 10) = (0, -54|108)$$

$$\begin{pmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & 0 & -54 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ 108 \end{pmatrix}$$

…AN EXAMPLE…

$$\begin{pmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & -15 & -9 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ 33 \end{pmatrix}$$
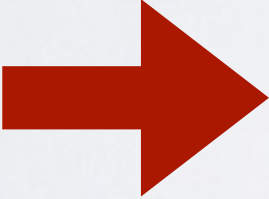
Gaussian Elimination

- $k°=2°$ step

  $j = 2,3$

  - $i = 3$: $m_{3,2} = \frac{-15}{-2} = \frac{15}{2}$

$$(-15, -9|33) - \frac{15}{2}(-2, 6| - 10) = (0, -54|108)$$

$$\begin{pmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & 0 & -54 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ 108 \end{pmatrix}$$

$$A^{(2)} = \begin{pmatrix} a_{1,1}^{(2)} & a_{1,2}^{(2)} & a_{1,3}^{(2)} \\ 0 & a_{2,2}^{(2)} & a_{3,2}^{(2)} \\ 0 & 0 & a_{3,3}^{(2)} \end{pmatrix} \qquad b^{(2)} = \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix}$$

…AN EXAMPLE…

$$\begin{pmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & -15 & -9 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ 33 \end{pmatrix}$$
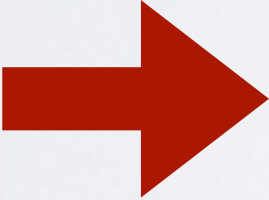
Gaussian Elimination

- k°=2° step

  - i = 3: $m_{3,2} = \frac{-15}{-2} = \frac{15}{2}$

j = 2,3

$$(-15, -9|33) - \frac{15}{2}(-2, 6| -10) = (0, -54|108)$$

$$\begin{pmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & 0 & -54 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ 108 \end{pmatrix}$$

The diagonal entries of the resulting matrix are called **pivot**.

…AN EXAMPLE

$$\begin{pmatrix} 1 & -3 & 1 \\ 0 & -2 & 6 \\ 0 & 0 & -54 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ 108 \end{pmatrix}$$

Backward substitution

$$x_3 = z = -\frac{108}{54} = -2$$

$$x_2 = y = -\frac{1}{2}(-10 - 6 \cdot (-2)) = -1$$

$$x_1 = x = (4 - 3 + 2) = 3$$

So the solution is: $(x, y, z)^T = (3, -1, -2)^T$

LU DECOMPOSITION

It is the matricial form of the Gaussian elimination.

It consists in writing the matrix A of the linear system Ax=b as the product of a lower and a upper triangular matrix, A=LU.

U is the upper triangular matrix obtained by the standard Gaussian elimination. L is the lower triangular matrix whose entries are the multipliers found during the Gaussian elimination, and whose diagonal entries are all 1.

$$
\begin{pmatrix}
a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
a_{2,1} & a_{2,2} & \cdots & a_{2,1} \\
\vdots & \vdots & \vdots & \vdots \\
a_{n,1} & a_{n,2} & \cdots & a_{n,n}
\end{pmatrix}
=
\begin{pmatrix}
1 & 0 & \cdots & 0 \\
m_{2,1} & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
m_{n,1} & m_{n,2} & \cdots & 1
\end{pmatrix}
\begin{pmatrix}
a_{1,1}^{(k)} & a_{1,2}^{(k)} & \cdots & a_{1,n}^{(k)} \\
0 & a_{2,2}^{(k)} & \cdots & a_{2,n}^{(k)} \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & a_{n,n}^{(k)}
\end{pmatrix}
$$

A   L   U

LU DECOMPOSITION

Once the LU decomposition of A is computed, one may rewrite the original system as follows:

$$Ax = b \implies LUx = b$$

In order to find the solution x, first solve $Ly = b$ using the forward substitution;

Afterwards find the solution x by solving $Ux = y$ with the forward substitution.

Notice that:
1. The Gaussian elimination is not applied to the vector of known terms, b;
2. Memory can be spared storing L and U in only one matrix.

FILL-IN

If we have a sparse matrix, the Gaussian elimination may destroy its sparsity.

$$
\begin{pmatrix}
1 & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\
\frac{1}{10} & 1 & 0 & 0 & 0 \\
\frac{1}{10} & 0 & 1 & 0 & 0 \\
\frac{1}{10} & 0 & 0 & 1 & 0 \\
\frac{1}{10} & 0 & 0 & 0 & 1
\end{pmatrix}
\xrightarrow{\text{LU factorization:}}
\begin{pmatrix}
1 & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\
\frac{1}{10} & \frac{99}{100} & -\frac{1}{100} & -\frac{1}{100} & -\frac{1}{100} \\
\frac{1}{10} & -\frac{1}{99} & \frac{98}{99} & -\frac{1}{99} & -\frac{1}{99} \\
\frac{1}{10} & -\frac{1}{99} & -\frac{1}{98} & \frac{97}{98} & -\frac{1}{98} \\
\frac{1}{10} & -\frac{1}{99} & -\frac{1}{98} & -\frac{1}{97} & \frac{96}{97}
\end{pmatrix}
$$

After the LU factorization, the matrix is no more sparse (**fill-in**).

FILL-IN

By swapping the first and the last row and the first and the last column, this phenomenon doesn't verify anymore.

$$
\begin{pmatrix}
1 & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\
\frac{1}{10} & 1 & 0 & 0 & 0 \\
\frac{1}{10} & 0 & 1 & 0 & 0 \\
\frac{1}{10} & 0 & 0 & 1 & 0 \\
\frac{1}{10} & 0 & 0 & 0 & 1
\end{pmatrix}
$$

Swap $\longrightarrow$

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & \frac{1}{10} \\
0 & 1 & 0 & 0 & \frac{1}{10} \\
0 & 0 & 1 & 0 & \frac{1}{10} \\
0 & 0 & 0 & 1 & \frac{1}{10} \\
1 & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10}
\end{pmatrix}
\longrightarrow
$$

LU factorization $\longrightarrow$

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & \frac{1}{10} \\
0 & 1 & 0 & 0 & \frac{1}{10} \\
0 & 0 & 1 & 0 & \frac{1}{10} \\
0 & 0 & 0 & 1 & \frac{1}{10} \\
1 & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & -\frac{3}{100}
\end{pmatrix}
$$

PIVOTING

A problem may occur when computing the multipliers:

$$m_{i,k} = \frac{a_{i,k}}{a_{k,k}}$$

What if $a_{k,k}=/\approx0$?

Recall that swapping two rows in a linear system doesn't effect the solution space (OP1). Hence one can swap the k° row with a row of index i > k such that $a_{i,k}$!=0 (**pivoting**).

In order to avoid divisions by zero, add the following control lines before computing the multiplier:
if $a_{k,k}$ == 0
    search in A(k+1:n,k) an entry $a_{r,k}$!=0 and swap r° and k° rows.

SOME EXAMPLES

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 3 \\ 3 & 1 & 3 \end{pmatrix} \qquad \sim \qquad \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 1 \\ 0 & -5 & -6 \end{pmatrix}$$

$m_{2,1} = 2 \quad row_2 - 2row_1$

$m_{3,1} = 3 \quad row_3 - 3row_1$

$m_{3,2} = \frac{-5}{0}$

Swap rows 2 and 3

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & -5 & -6 \\ 0 & 0 & 1 \end{pmatrix}$$

PIVOTING

small pivot

Well-known fact: **computers can't store all numbers** (irrational numbers, extremely low and extremely great numbers, rational numbers with many decimal digits…) and so different techniques to approximate numbers are used.

- **Most significant digit** of a number: first nonzero digit (from left).
- **Least significant digit** of a number: if the number is an integer, it is the last nonzero digit. Else it is its last digit, even though that is a 0.

PIVOTING

Well-known fact: **computers can't store all numbers** (irrational numbers, extremely low and extremely great numbers, rational numbers with many decimal digits…) and so different techniques to approximate numbers are used.

- **Most significant digit** of a number: first nonzero digit (from left).
- **Least significant digit** of a number: if the number is an integer, it is the last nonzero digit. Else it is its last digit, even though that is a 0.

Examples: 100200; 1234; 12340; 0.00102; 0.10000.

small pivot

Extreme situation: assume that the maximum number of digits that a computer can store is 2.

$$(A|b) = \begin{pmatrix} 0.001 & 1 & 3 \\ 1 & 2 & 5 \end{pmatrix}$$

$$m_{2,1} = \frac{1}{0.001} = 1000$$

SOME EXAMPLES

Extreme situation: assume that the maximum number of digits that a computer can store is 2.

$$(A|b) = \begin{pmatrix} 0.001 & 1 & 3 \\ 1 & 2 & 5 \end{pmatrix}$$

$$m_{2,1} = \frac{1}{0.001} = 1000$$

$$\begin{pmatrix} 0.001 & 1 & 3 \\ 0 & -998 & -2995 \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3.001 \end{pmatrix}$$

SOME EXAMPLES

Extreme situation: assume that the maximum number of digits that a computer can store is 2.

$$(A|b) = \begin{pmatrix} 0.001 & 1 & 3 \\ 1 & 2 & 5 \end{pmatrix}$$

$$m_{2,1} = \frac{1}{0.001} = 1000$$

$$\begin{pmatrix} 0.001 & 1 & 3 \\ 0 & \boxed{-998} & \boxed{-2995} \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3.001 \end{pmatrix}$$ 3 and 4 significant digits!

SOME EXAMPLES

Extreme situation: assume that the maximum number of digits that a computer can store is 2.

$$(A|b) = \begin{pmatrix} 0.001 & 1 & 3 \\ 1 & 2 & 5 \end{pmatrix}$$

$$m_{2,1} = \frac{1}{0.001} = 1000$$

$$\begin{pmatrix} 0.001 & 1 & 3 \\ 0 & -998 & -2995 \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3.001 \end{pmatrix}$$

$$\begin{pmatrix} 0.001 & 1 & 3 \\ 0 & -1000 & -3000 \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \end{pmatrix}$$

PIVOTING

Small pivoting can make solution very **unstable**, for this reason pivoting is used also when $a_{k,k}!=0$.

Partial Pivoting

at the k° step, find the maximum element (absolute value) in the sub column $a_{k:n}$ and swap the corresponding row and the k° row.

GEPP: before computing the multiplier, add:

find $|a_{r,k}| = \max_{k>=i>=n} |a_{i,k}|$ and swap rows k and r.

PIVOTING

Complete Pivoting

Search for the maximum element (absolute value) $|a_{r,s}|$ in the sub matrix $A_{k:n,k:n}$ and swap the k° row and the r° rows and the k° and the s° column.

GECP: before computing the multiplier, add:

find $|a_{r,s}| = \max_{\substack{k>=i>=n \\ k>=j>=n}} |a_{i,j}|$, swap rows k and r and columns k and s.

$$
\begin{pmatrix}
 & & \vdots & \\
 & -1 & 4 & 3 \\
\cdots & 5 & 0 & -9 \\
 & -1 & 3 & -1
\end{pmatrix}
\begin{pmatrix}
\vdots \\
x_{n-2} \\
x_{n-1} \\
x_n
\end{pmatrix}
=
\begin{pmatrix}
\vdots \\
1 \\
2 \\
3
\end{pmatrix}
$$

PIVOTING

Complete Pivoting

Search for the maximum element (absolute value) $|a_{r,s}|$ in the sub matrix $A_{k:n,k:n}$ and swap the k° row and the r° rows and the k° and the s° column.

GECP: before computing the multiplier, add:

find $|a_{r,s}| = \max_{k>=i>=n \atop k>=j>=n} |a_{i,j}|$, swap rows k and r and columns k and s.

$$\begin{pmatrix} & & \vdots & \\ & -1 & 4 & 3 \\ \cdots & 5 & 0 & -9 \\ & -1 & 3 & -1 \end{pmatrix} \begin{pmatrix} \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} \vdots \\ 1 \\ 2 \\ 3 \end{pmatrix} \Rightarrow \begin{pmatrix} & & \vdots & \\ & -9 & 0 & 5 \\ \cdots & 3 & 4 & -1 \\ & -1 & 3 & -1 \end{pmatrix} \begin{pmatrix} \vdots \\ x_n \\ x_{n-1} \\ x_{n-2} \end{pmatrix} = \begin{pmatrix} \vdots \\ 2 \\ 1 \\ 3 \end{pmatrix}$$

Swap rows n-2 and n-1
Swap columns n and n-2

When you swap two columns you need to swap the corresponding variables as well!

PIVOTING

Rook Pivoting

Search in the sub matrix $A_{k:n,k:n}$ for an element that has maximum absolute value both in its row and in its column

GERP: before computing the multiplier, add:

find $|a_{r,s}|=\max_{k>=i>=n}|a_{i,s}| = \max_{k>=j>=n}|a_{r,j}|$, swap rows k and r and columns k and s.

$$\begin{pmatrix} & & \vdots & \\ & -1 & 4 & 3 \\ \cdots & 5 & 0 & -9 \\ & -1 & 3 & -1 \end{pmatrix} \begin{pmatrix} \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} \vdots \\ 1 \\ 2 \\ 3 \end{pmatrix}$$

PIVOTING

Search in the sub matrix $A_{k:n,k:n}$ for an element that has maximum absolute value both in its row and in its column

GERP: before computing the multiplier, add:

find $|a_{r,s}|=\max_{k>=i>=n}|a_{i,s}| = \max_{k>=j>=n}|a_{r,j}|$, swap rows k and r and columns k and s.

$$\left(\begin{array}{ccc} & \vdots & \\ -1 & 4 & 3 \\ \cdots \quad 5 & 0 & -9 \\ -1 & 3 & -1 \end{array} \right) \left(\begin{array}{c} \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{array} \right) = \left(\begin{array}{c} \vdots \\ 1 \\ 2 \\ 3 \end{array} \right) \quad \Longrightarrow \quad \left(\begin{array}{ccc} & \vdots & \\ 4 & -1 & 3 \\ \cdots \quad 0 & 4 & -9 \\ 3 & -1 & -1 \end{array} \right) \left(\begin{array}{c} \vdots \\ x_{n-1} \\ x_{n-2} \\ x_n \end{array} \right) = \left(\begin{array}{c} \vdots \\ 1 \\ 2 \\ 3 \end{array} \right)$$

swap columns n-1 and n-2

When you swap two columns you need to swap the corresponding variables as well!

SOME EXAMPLES

small pivot

partial pivoting

$$(A|b) = \begin{pmatrix} 0.001 & 1 & 3 \\ 1 & 2 & 5 \end{pmatrix} \quad\Rightarrow\quad \begin{pmatrix} 1 & 2 & 5 \\ 0.001 & 1 & 3 \end{pmatrix} \quad\Rightarrow$$

$$m_{2,1} = \frac{0.001}{1} = 0.001$$

$$\Rightarrow \begin{pmatrix} 1 & 2 & 5 \\ 0 & 0.998 & 2.995 \end{pmatrix} \qquad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3.001 \end{pmatrix}$$

SOME EXAMPLES

small pivot

partial pivoting

$$(A|b) = \begin{pmatrix} 0.001 & 1 & 3 \\ 1 & 2 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 5 \\ 0.001 & 1 & 3 \end{pmatrix}$$

$$m_{2,1} = \frac{0.001}{1} = 0.001$$

$$\begin{pmatrix} 1 & 2 & 5 \\ 0 & 0.998 & 2.995 \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3.001 \end{pmatrix}$$

SOME EXAMPLES

partial pivoting

$$(A|b) = \begin{pmatrix} 0.001 & 1 & 3 \\ 1 & 2 & 5 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 2 & 5 \\ 0.001 & 1 & 3 \end{pmatrix} \Rightarrow$$

$$m_{2,1} = \frac{0.001}{1} = 0.001$$

$$\begin{pmatrix} 1 & 2 & 5 \\ 0 & 0.998 & 2.995 \end{pmatrix} \qquad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3.001 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 5 \\ 0 & 1 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 3 \end{pmatrix}$$

PIVOTING

Partial, complete and rook pivoting ensure **stability** since multipliers are always <=1 and **avoid divisions by zero**.

If A is nonsingular, pivoting **always succeeds**, meaning that one shall find a non zero element in the sub column and in the sub matrix at every step.

How expensive is pivoting?

Partial pivoting   **Complete pivoting**   **Rook pivoting**

$O(n^2)$ $O(n^3)$ $O(n^2) \leq T(n) \leq O(n^3)$

+ **communication time** (time required for moving data)

PIVOTING

Pivoting may be necessary when solving linear systems. Nevertheless it is **expensive**. When pivoting is applied, the rows and the columns to swap need to be **copied** into temporary arrays, which takes long time.

How to solve this? Either devising faster pivoting algorithms, or trying to **avoid** pivoting.

We say that a matrix is **Gauss-eliminable** if the necessity of pivoting is never encountered during the execution of the Gaussian elimination.

RBT-DEFINITIONS

Linear transformations applied to the coefficient matrix that output with high probability a Gauss-eliminable matrix.

Butterfly matrix:

$$B^{<n>} = \frac{1}{\sqrt{2}} \begin{pmatrix} R_0 & R_1 \\ R_0 & -R_1 \end{pmatrix}$$

where R_0 and R_1 are **diagonal** and **nonsingular** matrices of size n/2.

Direct sum of matrices:

$$A_{m \times n}, B_{p \times q} \qquad C_{(m+p) \times (n+q)} = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} = A \oplus B$$

RBT-DEFINITIONS

Recursive butterfly matrix:

$$U^{<n>} = (U_0^{<\frac{n}{2}>} \oplus U_1^{<\frac{n}{2}>})B^{<n>} = \begin{pmatrix} U_0^{<\frac{n}{2}>} & 0 \\ 0 & U_1^{<\frac{n}{2}>} \end{pmatrix} B^{<n>}$$

$$U^{<1>} = u$$

where B$^{<n>}$ is a butterfly matrix of dimension n.

…Deriving compact formulation solving the recursive equation:

$$U^{<n>} = \bigoplus_{i=0}^{n-1} U_i^{<1>} \cdot \prod_{i=0}^{\log(n)-1} \left(\bigoplus_{j=0}^{2^i-1} B_j^{<\frac{n}{2^i}>} \right)$$

What can we say about n?

RBT-DEFINITIONS

Recursive butterfly matrix:

$$U^{<n>} = (U_0^{<\frac{n}{2}>} \oplus U_1^{<\frac{n}{2}>})B^{<n>} = \begin{pmatrix} U_0^{<\frac{n}{2}>} & 0 \\ 0 & U_1^{<\frac{n}{2}>} \end{pmatrix} B^{<n>}$$

$$U^{<1>} = u$$

where B$^{<n>}$ is a butterfly matrix of dimension n.

…Deriving compact formulation solving the recursive equation:

$$U^{<n>} = \bigoplus_{i=0}^{n-1} U_i^{<1>} \cdot \prod_{i=0}^{\log(n)-1} \left(\bigoplus_{j=0}^{2^i-1} B_j^{<\frac{n}{2^i}>} \right)$$

What can we say about n?

…n must be a power of 2!

RBT

Random butterfly transformation: $\tilde{A} = U^T A V$ where U and V are recursive random butterfly matrices.

Theorem: RBTs output Gauss-eliminable matrices with high probability.

RBT

Random butterfly transformation: $\tilde{A} = U^T AV$ where U and V are recursive random butterfly matrices.

Theorem: RBTs output Gauss-eliminable matrices with high probability.

Algorithm:

Given a linear system Ax=b:

- If size(A)=n is not a power of 2, augment A to the next most power of 2 by adding ones to the diagonal and zeros in the extra entries.
- Generate recursive butterfly matrices U and V with random entries in $[e^{-1/20}, e^{1/20}]$.
- Apply GE without pivoting (GENP) on $\tilde{A}y = U^T b$ ($U^T AVy = U^T b$) and solve the resulting upper-triangular linear system (backward substitution).
- Output the solution $x|_{1:n} = Vy|_{1:n}$.

GAUSS-ELIMINABLE MATRICES

The following classes of matrices are Gauss-eliminable:

Strictly diagonally dominant matrices

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

Positive-definite matrices

In the **complex** field:

$$\Re(x^* A x) > 0 \quad \forall x \in \mathbb{C}^n \neq 0$$

where x^* is the conjugate transpose.

In the **real** field:

$$x^T A x > 0 \quad \forall x \in \mathbb{R}^n \neq 0$$

POSITIVE-DEFINITE MATRICES

Since positive-definite matrices are Gauss-eliminable, another way to avoid pivoting is transforming the coefficient matrix A into a positive-definite matrix.

Two methods:

- Multiply A by its **transpose**.

Given A (even non square), A^TA and AA^T are positive-semi definite.

- Compute the **exponential** of A.

Given a square and **symmetric** matrix A, exp(A) is positive-definite.

TRANSPOSE METHOD

Given a linear system Ax=b:

- Compute $A^\top Ax = A^\top b$.

- Apply GENP to the resulting linear system.

- Solve the resulting upper-triangular linear system with backward substitution.

EXPONENTIAL MATRIX

If A is a square matrix, then the matrix exponential of A is defined as:

$$exp(A) \equiv e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}$$

How to compute the exponential matrix?

EXPONENTIAL MATRIX

If A is a square matrix, then the matrix exponential of A is defined as:

$$exp(A) \equiv e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}$$

How to compute the exponential matrix?

Analitically:

- Diagonalize A, e.g.:
    - A. Compute the spectrum of A by solving the characteristic polynomial: det(A-λ I) = 0.
    - B. Compute the eigenvectors of A, that are the columns of matrix P.
    - C. Compute the diagonal matrix: D =P^{-1}AP. (if A is symmetric, P is **orthogonal**).

- exp(A)=Pexp(D)P^{-1}. Notice that the exp(D)=exp(diag(λ_1,…,λ_n))= =diag(e$^{\lambda_1}$,…,e$^{\lambda_n}$).

EXPONENTIAL MATRIX

If A is a square matrix, then the matrix exponential of A is defined as:

$$exp(A) \equiv e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}$$

How to compute the exponential matrix?

Matlab:

- Follow the analytic approach using the Matlab function **eig** or other commands to find eigenvalues and eigenvectors, or.
- Compute directly the exponential using the Matlab function **exp**.

EXPONENTIAL METHOD

Given a linear system Ax=b where A is symmetric:

- Compute the spectrum of A and its eigenvectors and store them column-wise in a matrix P.

- Compute $\exp(D)=\text{diag}(e^{\lambda_1},\ldots,e^{\lambda_n})$ and $\exp(A)=P\exp(D)P^{-1}=P\exp(D)P^T$.

- Compute $F = \text{diag}(\lambda_1/e^{\lambda_1},\ldots,\lambda_n/e^{\lambda_n})$ and $b' = PFP^Tb$.

- Apply GENP on $\exp(A)x=b'$ and solve the resulting linear system.

RBT VS. POSITIVE-DEFINITE METHODS

RBT, Transpose and Exponential methods are to be applied on **dense** matrices.
The characteristics of positive-definite methods are the following:

- No need to augment the input linear system (while it has to be augmented if its size is not a power of 2 in RBT methods).
- They generate positive-definite matrices (we will see how this is advantageous).
- They are less stable than RBT. Trick: multiply the matrix by a constant < 1 before applying the preconditioners.
- Computing the matrix exponential is expensive, but it is parallelizable and Matlab has optimized functions to compute it.

EXPERIMENTS
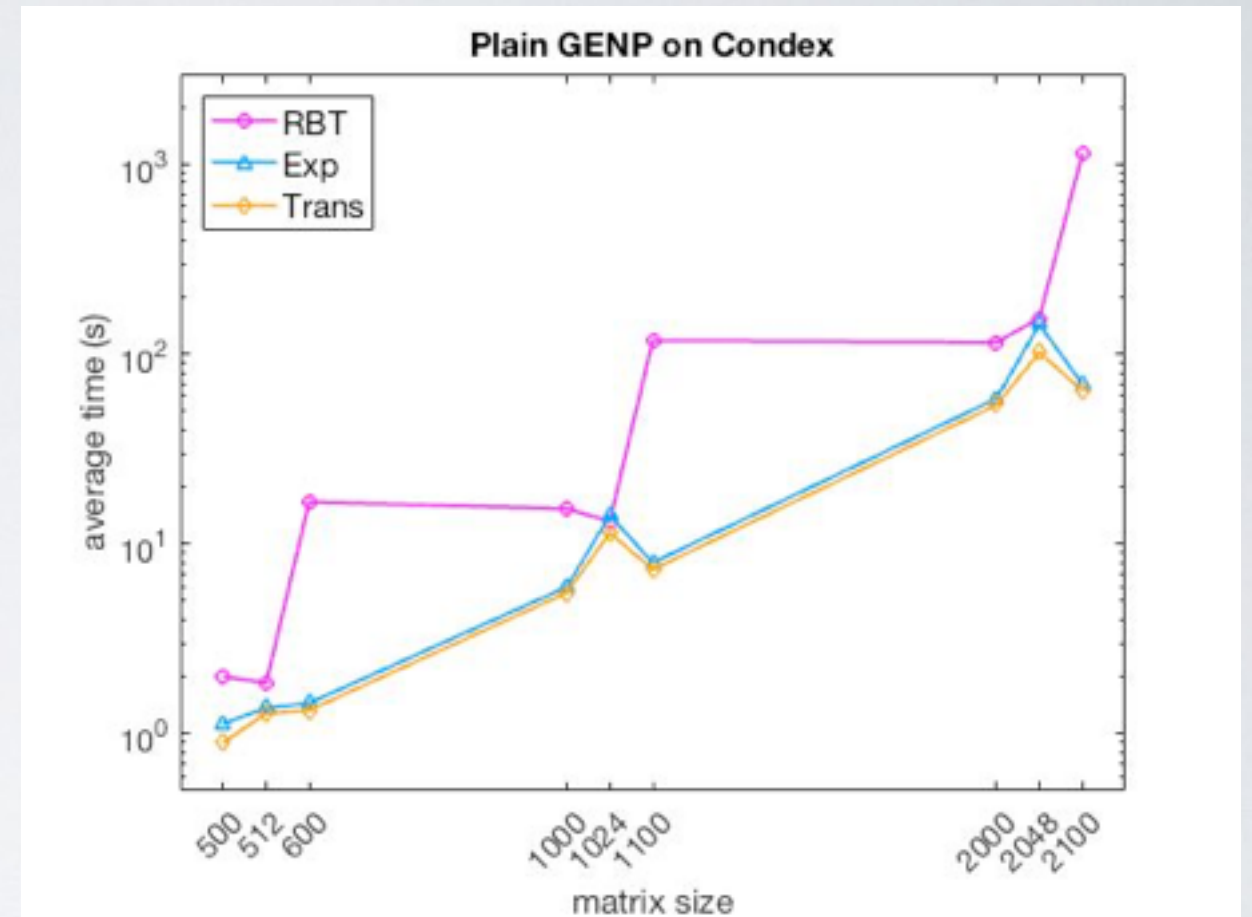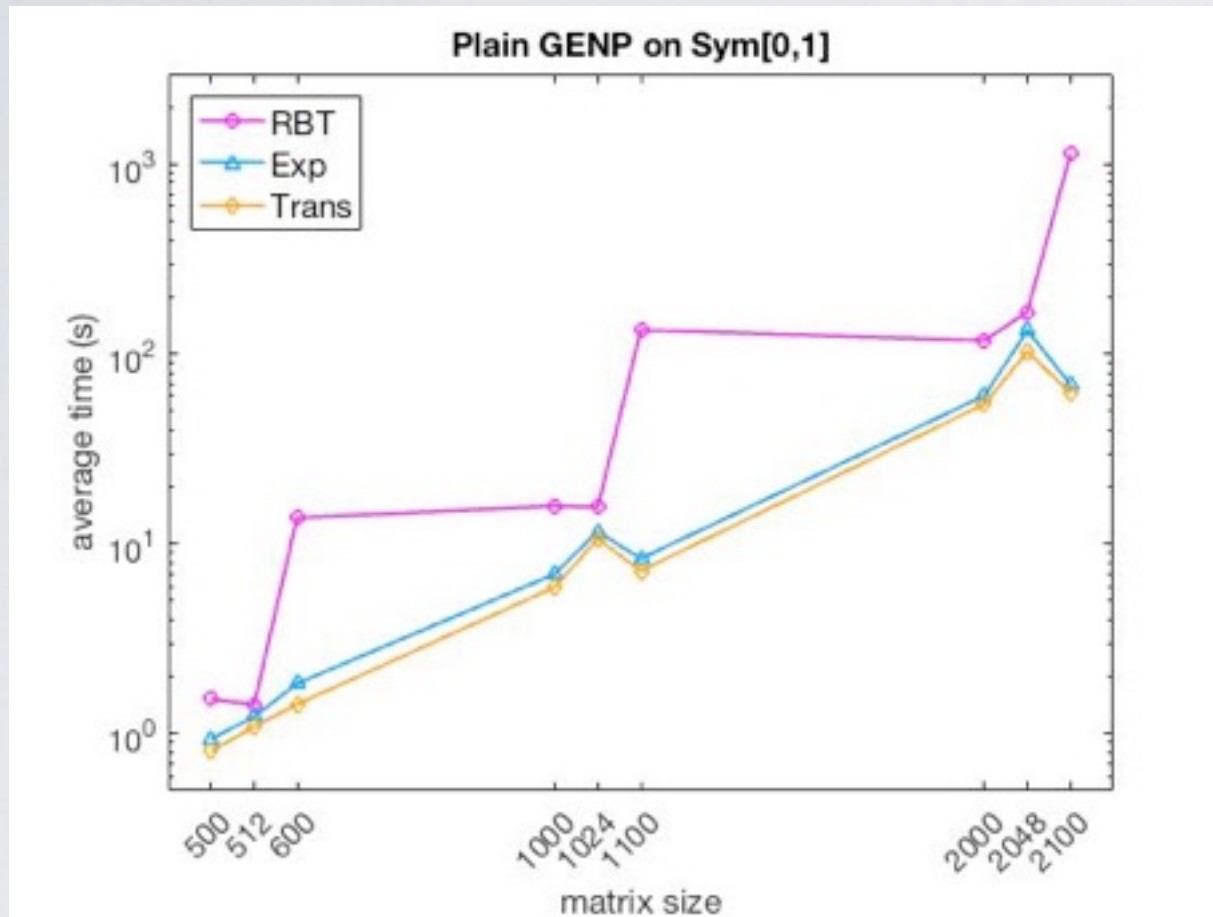
- matrix sizes: 500, 512, 600, 1000, 1024, 1100, 2000, 2048, 2100.
- RBT trend is stair-like:

RBTtime$_{600}$ ~ RBTtime$_{1024}$, RBTtime$_{1100}$ ~ RBTtime$_{2100}$.
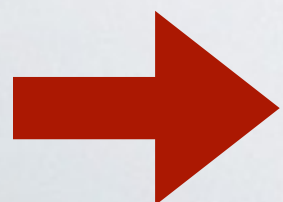
What are those time peaks for Exp and Trans in correspondence of matrix sizes 2^k??

CACHE THRASHING

Eviction of useful data from the cache, either because the useful data **fills the cache**, so portions of data must be loaded in the cache in a different moment, or because useful datas have the same address in the cache (direct addressing) and so they are loaded **in the same location**.

Consider allocating a matrix of size $2^k \times 2^k$. If k is sufficiently large, the rows or the columns of the matrix refer to the same location in the cache, hence every time one of them is needed, it evicts the array that is already in the cache. This implies a lot of movement due to repeatedly loading and unloading of the cache, even if this is not full. Notice that this conflict between addresses doesn't occur if the matrix size is augmented even just by 1.
Solution: breaking the matrices into blocks of size != power of 2.

NEW EXPERIMENTS
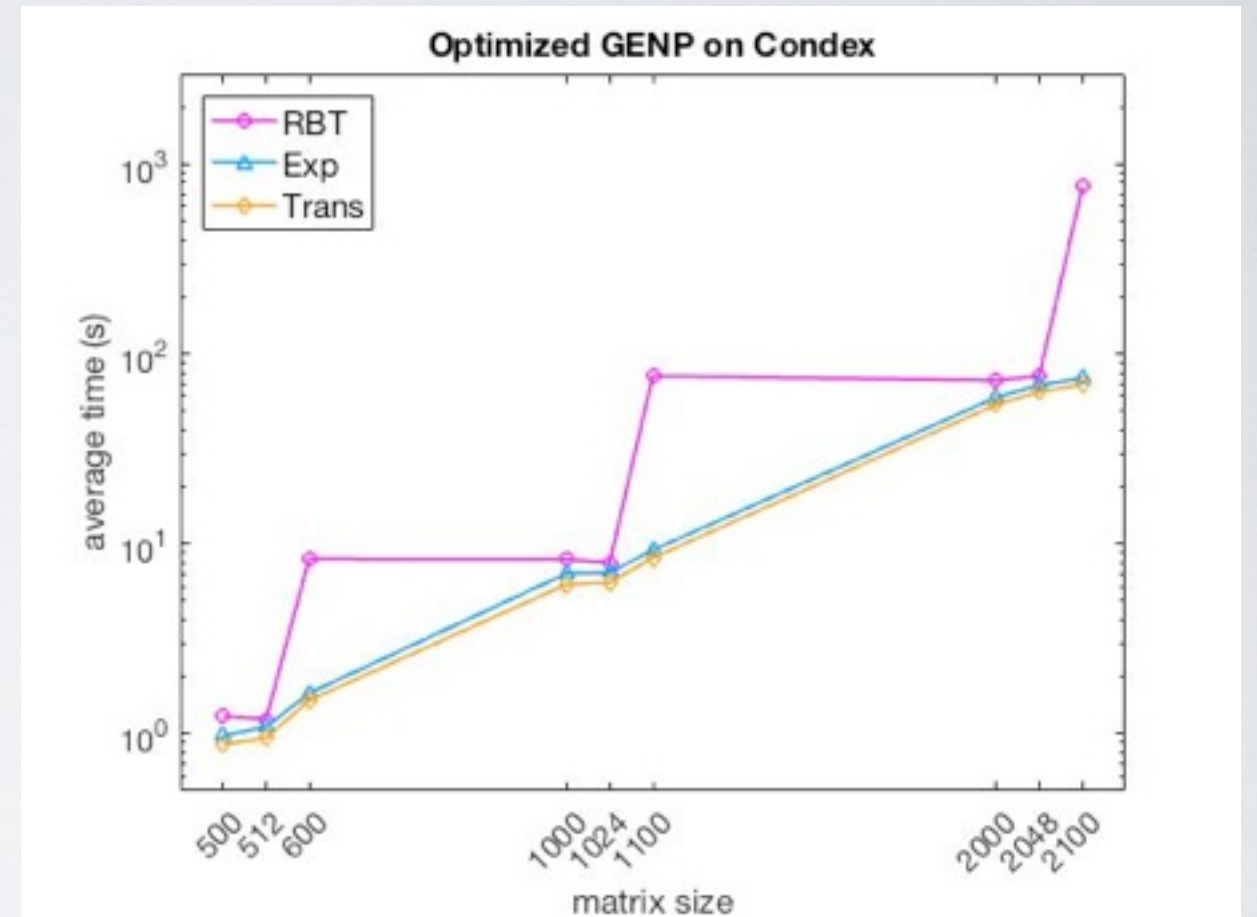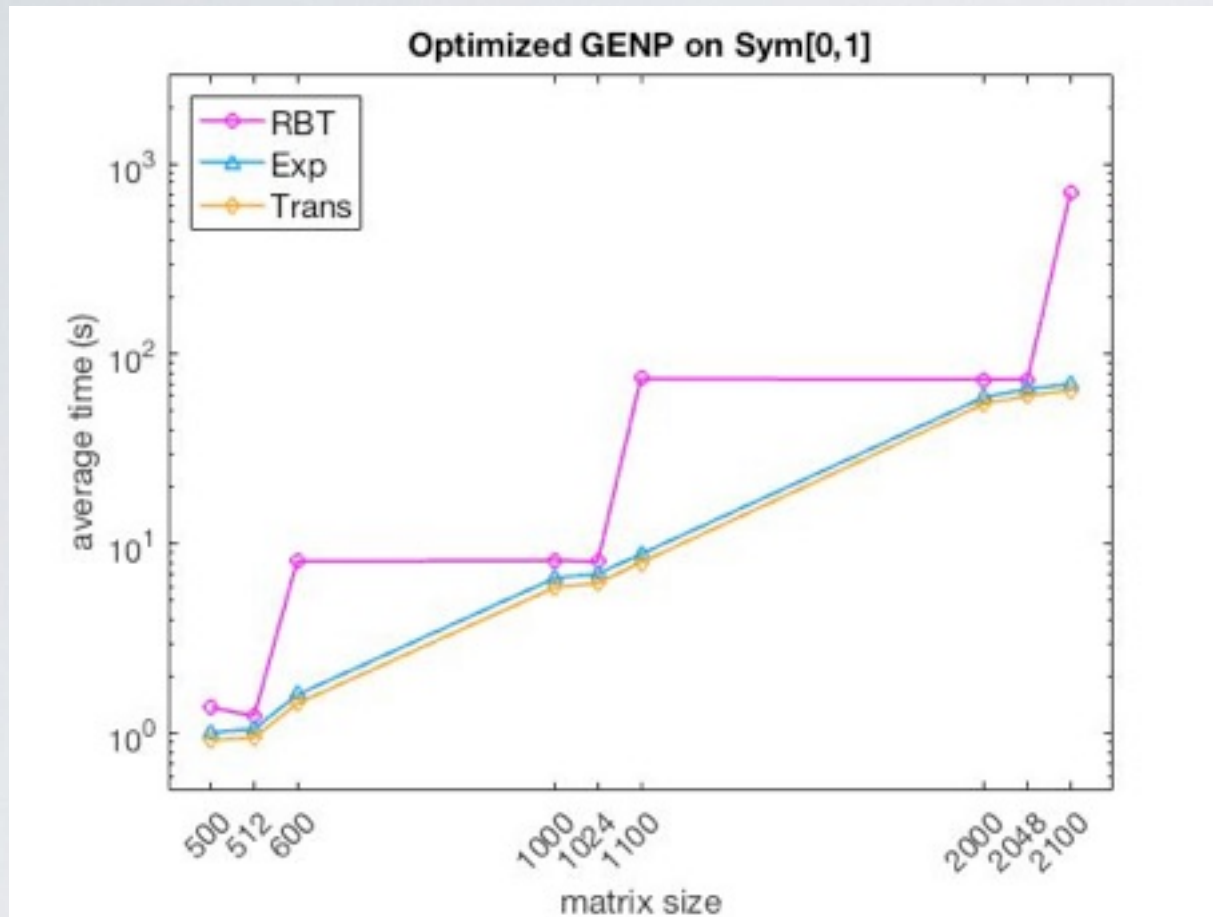
- Time peaks are completely flattered at the expense of a small general overhead (for Exp and Trans).
- Consistent time improvement for RBT, for all sizes: this is because RBT **always** augments the input matrix to a power of 2.

MULTI-THREADED IMPLEMENTATION - OPENMP

LU Matlab function is still faster.

Multi-threated implementation results with OpenMP on 14 matrices of size 10000:

- Trans between 24% and 40% faster than GEPP
- The percentage of time spent for computing A^TA and A^Tb oscillates around 36.5% of the total time for Trans.
- Errors achieved by GEPP are lower.