Homework nr. 3

The files (a_i.txt, b_i.txt, i=1,...,5) contain 5 linear systems with sparse matrix (with *'few'* non-zero elements, $a_{ij} \neq 0$ and size n *'big'*), $Ax = b$, in the following way:

- For matrix A (files a_i.txt):
  - n system's size,
  - $a_{ij} \neq 0$, i, j - non-zero values from the sparse matrix $A \in \mathbb{R}^{n \times n}$, row index and column index of the correspunding non-zero element
- For vector b (files b_i.txt):
  - n vector's size,
  - b_i, $i = 1, 2, \ldots, n$, all the elements of vector $b \in \mathbb{R}^n$.

1. Using the attached files, read the size of the linear system, the vector b, the non-zero elements of matrix A, generate the necessary data structures for economically storing the sparse matrix. For matrix A use the sparse storing method described below. Assume that the non-zero elements are randomly located in the file. Verify that the diagonal elements of the matrix are non-zero. Use two methods for memorizing the sparse matrix, one that is described in this text and the second, of your choice. For both these methods, solve the below-described requirements. Other methods for storing sparse matrices can be found here. Consider the computation error $\epsilon = 10^{-p}$ as input.

2. Using the sparse storage for matrix A, approximate the solution of the linear system:
$$Ax = b \tag{1}$$
employing the Gauss-Seidel method. Display (also) the number of iterations performed until convergence.

3. Verify the correctness of the computed solution by displaying the norm:
$$\|Ax_{GS} - b\|$$
where x_{GS} is the approximate solution obtained using Gauss Seidel method.

4. In all computations that involve the elements of matrix A, use the sparse storage structures (do not declare a classical matrix).

5. When implementing the Gauss Seidel method use only one vector x_{GS}.

Bonus 25pt: compute the sum of two sparse matrices (a.txt, b.txt, aplusb.txt). Verify that the (computed) sum of matrices from files *a.txt* and *b.txt* is the matrix from file *aplusb.txt*. Two elements that have the same row and column indices (i, j) are considered equal if $|c_{ij} - d_{ij}| < \epsilon$. Use both methods for storing sparse matrices.

Remarks:

1) Don't use classically stored matrices for solving the above problem and don't use a function $val(i, j)$ that computes the value of a_{ij} the corresponding element in the matrix, for all pairs (i, j).

2) Implementing the sparse storage method described in this file is **mandatory** (not implementing it will be penalised).

3) If in the *a_i.txt* files, there are multiple entries with the same row and column indices:

$$\begin{array}{c} val_1 \quad , \quad i \quad , \quad j \\ \vdots \\ val_2 \quad , \quad i \quad , \quad j \\ \vdots \\ val_k \quad , \quad i \quad , \quad j \end{array}$$

this situation has the following meaning:

$$a_{ij} = val_1 + val_2 + \cdots + val_k.$$

Sparse matrix storing method

A sparse vector is a vector that has 'few' non-zero elements. Such a vector can be stored economically in a data structure that will keep only the non-zero values and the position in the vector of the respective value:

$$\{(val \neq 0, i); x_i = val\}$$

A sparse matrix can be economically stored in the following way:

– a vector $d \in \mathbb{R}^n$ is used for storing the diagonal elements of the matrix, $d_i = a_{ii}$, $i = 1, \ldots, n$

– a vector of sparsely stored vectors: the non-zero elements of each row except the diagonal one are stored in a sparse vector.

Iterative methods for solving sparse linear systems

Assume that the matrix A is nonsingular, i.e. $\det A \neq 0$. Denote the exact solution of system (1) by x^*:

$$x^* = A^{-1}b$$

The iterative methods for solving linear systems were developed for *'large'* systems (n *'large'*), with sparse matrix A (a matrix that has *'few'* non-zero elements $a_{ij} \neq 0$). The iterative methods do not change the matrix A (as it happens in Gaussian elimination or in LU decompositions or in QR factorizations), the non-zero elements of the matrix are employed for approximating the exact solution x^*. For sparse matrices, special storing methods are employed.

For approximating the solution x^*, a sequence of real vectors $\{x^{(k)}\} \subseteq \mathbb{R}^n$ is computed. If certain conditions are fulfilled, this sequence converges to the exact solution x^* of system (1):

$$x^{(k)} \longrightarrow x^* \quad , \quad k \longrightarrow \infty.$$

The first vector of the sequence, $x^{(0)}$, is usually set to 0:

$$x_i^{(0)} = 0 \ , \quad i = 1, \ldots, n$$

When the sequence $\{x^{(k)}\}$ converges, the limit is x^* the exact solution of system (1).

Gauss-Seidel Method

In order to be able to apply this method all the diagonal elements of matrix A must be non-zero:

$$a_{ii} \neq 0 \quad \forall i = 1, \ldots, n$$

When reading the matrix from one of the posted files, verify that all the diagonal elements of the matrix are non-zero ($|a_{ii}| > \epsilon \, \forall i$). If there exists a

zero diagonal element, the system cannot be solved using the Gauss-Seidel iterative method. The sequence of vectors generated by the Gauss-Seidel iterative method, is computed with the following relation:

$$\begin{aligned} x_i^{(k+1)} &= \frac{\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}\right)}{a_{ii}} \quad , \quad i = 1, \ldots, n \\ &= \frac{\left(b_i - \sum_{\substack{\text{non-zero elements on row } i \\ \text{except the diagonal element}}} val * x^{(?)}_{column\ index}\right)}{\text{the value of the diagonal element on row } i} \end{aligned} \tag{2}$$

The above formula must be adapted to the new way of storing the sparse matrix A. In the above sums for computing the i-th component of a vector, one needs only the non-zero elements a_{ij} from row i. For fast computing the i-th component of vector $x(k+1)$ we need to have easy access to the non-zero elements of matrix A from row i, that's why in the sparse storing methods the elements are memorized accordingly. If, for example, the matrix A is row diagonally dominant, that is:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \text{for all } i = 1, \ldots, n$$

the sequence $\{x^{(k)}\}$ computed with the Gauss-Seidel algorithm, converges to solution x^* (but it is not the only situation when the sequence converges).

For all iterative methods that solve linear systems, the convergence or divergence of sequence $\{x^{(k)}\}$ doesn't depend on the choice of the first vector $x^{(0)}$ of the sequence.

In order to get an approximation for solution x^* one must compute an element $x^{(k)}$ of the sequence of vectors, for k sufficiently large. If the difference between two consecutive elements of the sequence $\{x^{(k)}\}$ becomes sufficiently *'small'*, then the last computed vector is *'near'* the exact solution:

$$\begin{aligned} \|x^{(k+1)} - x^{(k)}\| \leq \epsilon \quad &\Rightarrow \quad \|x^{(k+1)} - x^*\| \leq c\epsilon \ , \ c \in \mathbb{R}_+ \\ &\Rightarrow \quad x^{(k+1)} \approx x^*. \end{aligned} \tag{3}$$

It is not necessary to store all the vectors of the sequence $\{x^{(k)}\}$, we only need the last two computed elements, and for approximating the solution we need the vector that satisfies relation (3), $\|x^{(k)} - x^{(k-1)}\| \leq \epsilon$.

In your program, you can use only two vectors:

$$x^c \text{ for vector } x^{(k+1)} \text{ and } x^p \text{ for vector } x^{(k)}.$$

The Gauss-Seidel method can be implemented by only using one vector for all the computations.

$$x_{GS} = x^c = x^p.$$

When using only one vector, computing formula (2) and the norm computation $\|x^{(k+1)} - x^{(k)}\| = \|x^c - x^p\|$ must be done in the same *for* loop.

Iterative method for solving a linear system (with 2 vectors!)

$x^c = x^p = 0$;

$k = 0$;

do

{

$x^p = x^c$;

compute new x^c using x^p (with formula (2)) ;

compute new $\Delta x = \|x^c - x^p\|$;

$k = k + 1$;

}

while ($\Delta x \geq \epsilon$ and $k \leq k_{max}$ and $\Delta x \leq 10^{10}$)

if ($\Delta x < \epsilon$)

$x^c \approx x^*$; // x^c is the approximation of the exact solution

else ***'divergence'*** ;

($k_{max} = 10000$, $\epsilon = 10^{-p}$, $p \in \{5, 6, 7, 8, 9\}$)

Example

Matricea:

$$A = \begin{pmatrix} 102.5 & 0.0 & 2.5 & 0.0 & 0.0 \\ 3.5 & 104.88 & 1.05 & 0.0 & 0.33 \\ 0.0 & 0.0 & 100.0 & 0.0 & 0.0 \\ 0.0 & 1.3 & 0.0 & 101.3 & 0.0 \\ 0.73 & 0.0 & 0.0 & 1.5 & 102.23 \end{pmatrix}$$

se poate memora economic astfel:

$$d = (102.5, 104.88, 100.0, 101.3, 102.23)$$

$$\begin{array}{ll} \{ & \{(2.5, 2)\}, \\ & \{(3.5, 0), (0.33, 4), (1.05, 2)\}, \\ & \{null\}, \\ & \{(1.3, 1)\}, \\ & \{(1.5, 3), (0.73, 0)\} \quad \}. \end{array}$$

Presupunem că:

$$x^{(0)} = \begin{pmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \\ 5.0 \end{pmatrix} \quad , \quad b = \begin{pmatrix} 6.0 \\ 7.0 \\ 8.0 \\ 9.0 \\ 1.0 \end{pmatrix}$$

$x_0^{(1)}$ (varianta clasică)

$= (b_0 - a_{01}x_1^{(0)} - a_{02}x_2^{(0)} - a_{03}x_3^{(0)} - a_{04}x_4^{(0)})/a_{00} -$

$= (6.0 - 0.0 * 2.0 - 2.5 * 3.0 - 0.0 * 4.0 - 0.0 * 5.0)/102.5$

(varianta economică foloseşte doar elementele nenule de pe linia1)

$= (6.0 - 2.5 * 3.0)/102.5 = -0,01463414...$

$x_1^{(1)}$ (varianta clasică)

$= (b_1 - a_{10}x_0^{(1)} - a_{12}x_2^{(0)} - a_{23}x_3^{(0)} - a_{24}x_4^{(0)})/a_{11} -$

$= (7.0 - 3.5 * (-0.01463414...) - 1.05 * 3.0 - 0.0 * 4.0 - 0.33 * 5.0)/104.88$

(varianta economică foloseşte doar elementele nenule de pe linia a 2-a)

$= (7.0 - 1.05 * 3.0 - 3.5 * (-0.01463414...) - 0.33 * 5.0)/104.88 = 0.0214647...$