# Problem sheet 2

Joe & Víctor

Department of Aeronautics
Imperial College London

The idea of this problem sheet is to get familiar with the concept of *sparsivity* in matrices. As in the previous problem sheet, you should try to solve the exercises manually first (by hand), and then implement them in `Python`. You will find the solutions to the exercises in the GitHub repository https://github.com/victorballester7/sparseIMS/tree/main/src/solutions.

## Sparse matrices

A sparse matrix is a matrix in which many of the elements are zero. For example, the following matrix is sparse:

$$\begin{pmatrix} 0 & 0 & 3 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \end{pmatrix}.$$

Contrarily, we will talk about dense matrices when most of the elements are non-zero. For example, the following matrix is dense:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 0 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 0 & 23 & 24 & 25 \end{pmatrix}.$$

Of course, the concept of sparse matrix only makes sense for large matrices, since in small matrices (e.g. $2 \times 2$ or $3 \times 3$) it is unlikely to find "many" zero elements.

A natural concept that arises when talking about sparse matrices is the *sparsity* $s$ of a matrix, which is defined as the number of zero-valued elements divided by the total number of elements of a matrix (e.g. $n^2$ for an $n \times n$ matrix). Analogously, the *density* $\rho$ of a matrix is defined as the number of non-zero elements divided by the total number of elements. Note that $s + \rho = 1$.

Sparse matrices are very common in scientific computing. Because of this, specific algorithms and data structures have been developed to efficiently store and manipulate sparse matrices. Briefly speaking, these algorithms only store the non-zero elements of the matrix (e.g the value of the element and its position in the matrix), which saves memory and computational resources. Keeping this in mind, we expect to observe a reduction in computational time when working with sparse matrices compared to dense matrices, especially for very large sparse matrices. This is the main motivation for this problem sheet and for the future comparison with dense matrices when solving linear systems.

**Exercise 1.** Consider the matrices

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \end{pmatrix} + 6\mathbf{I}_5, \quad \mathbf{B} = \begin{pmatrix} 7 & 2 & 0 & \cdots & 0 \\ 3 & 7 & 2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 3 & 7 & 2 \\ 0 & \cdots & 0 & 3 & 7 \end{pmatrix} \in \mathbb{R}^{200\times 200}$$

where $\mathbf{I}_5$ denotes the $5 \times 5$ identity matrix, and the matrix $\mathbf{B}$ is of size $200 \times 200$.

1. Compute $s(\mathbf{A})$ $\rho(\mathbf{A})$, $s(\mathbf{B})$ and $\rho(\mathbf{B})$ and check that $s(\mathbf{A}) + \rho(\mathbf{A}) = s(\mathbf{B}) + \rho(\mathbf{B}) = 1$.

   *Note.* In Python, the matrix $\mathbf{A}$ can be defined as

   ```
   import numpy as np
   A = np.zeros((5, 5))
   ```

   and then filling in the non-zero elements manually as

   ```
   A[1, 0] = -1
   ...
   ```

   Remember that the identity matrix can be created using

   ```
   I = np.identity(5)
   ```

   For the matrix $\mathbf{B}$, the hint is that the following code

   ```
   n = 200
   k = 4
   diagK = np.diag(value * np.ones(n - k), k)
   ```

   will generate the matrix with the value `value` in the $k$-th diagonal (counting from the main diagonal, which is $k = 0$):

$$\begin{pmatrix} 0 & 0 & 0 & 0 & \texttt{value} & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \texttt{value} & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \texttt{value} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

   A negative value of $k$ will create the corresponding sub-diagonal. You can use this to create the diagonals of $\mathbf{B}$. Note the $n - k$ in the code above, which ensures that the total matrix is of size $n \times n$ (the $k$-th diagonal has $n - k$ elements).

   To compute the density of the matrices, you can use the following code:

   ```
   num_nonzero_A = np.count_nonzero(A)
   total_elements_A = A.size
   ```

## Generation of sparse matrices

**Exercise 2.** Our goal in this exercise is to create a function that generates an invertible random sparse matrix of given size $n$ and given density $\rho$. The main difficulty that one may find when implementing this function is to ensure that the generated matrix is invertible. An intuitive trick to ensure that a matrix is invertible is to make it diagonally dominant, which means intuitevely that the elements in the main diagonal are much larger than the other elements in the same row. An example of such a matrix is the matrix $\mathbf{A}$ in the previous exercise. In that case since a diagonal matrix (with non-zero elements in the diagonal) is always invertible, adding a "perturbation" to it will still result in an invertible matrix. The formal theorem that guarantees this is Greshgorin's circle theorem (https://en.wikipedia.org/wiki/Gershgorin_circle_theorem). Briefly:

**Theorem 1 (Gershgorin's circle theorem).** Let $\mathbf{M} = (m_{ij}) \in \mathbb{R}^{n\times n}$, e.g.

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{pmatrix}.$$

For $i = 1, \dots, n$, let

$$R_i = \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|$$

be the sum of the absolute values of the non-diagonal entries in row $i$, and define the Gershgorin discs

$$D_i = \text{disc in the complex plane with centre } a_{ii} \text{ and radius } R_i,$$

Then, every eigenvalue of $\mathbf{A}$ lies within at least one of the discs $D_i$.

By the previous theorem, if we ensure that for every row $i$ $|a_{ii}| > R_i$, then all the eigenvalues of the matrix will be non-zero, which guarantees that the matrix is invertible (check previous problem sheet).

1. Using the Greshgorin's circle theorem we will need to fill the diagonal elements of the matrix last. So if $\mathbf{A}$ is our final matrix of size $n \times n$, then we can decompose it as

   $$\mathbf{A} = \mathbf{M} + \mathbf{D},$$

   where $\mathbf{M}$ is a sparse matrix with zero elements in the diagonal and $\mathbf{D}$ is a diagonal matrix that will be filled at the end. In order for $\mathbf{A}$ to have density $\rho$, we will need to fill $\mathbf{M}$ with a certain number of elements resulting in $\mathbf{M}$ having density $\rho' < \rho$. Find the (rounded or approximated) number $m$ of non-zero and non-diagonal elements that $\mathbf{M}$ must have in order to make $\mathbf{A}$ a matrix of density $\rho$.

2. Generate vectors `rows` and `cols` of size $m$ that contain the row and column indices of the non-zero non-diagonal elements of the matrix. The coordinates of both vectors must be in the set $\{0, \dots, n-1\}$ and there cannot be any repeated coordinate pair $(\texttt{rows}[i], \texttt{cols}[i])$. You can use the function `numpy.random.randint(low, high, size)` to generate random integers in the range $[\texttt{low}, \texttt{high})$.

   *Note.* I was not able to find a clever way of doing this avoiding while or for loops. But `chatGPT` helped me with that. Damn, he is clever! Have a look at the function `sample_offdiagonal()` in the solutions of this exercise.

3. Generate $m$ random values within the interval $(-a, a)$, with $a \in \mathbb{R}$. Those will be the values of the non-zero non-diagonal elements of the matrix $\mathbf{M}$. Store those values in a vector `values` of size $m$.

   Now we can create our matrix using

   ```
   M = np.zeros((n, n))
   M[rows, cols] = values
   ```

4. Row by row, compute the sum of the absolute values of the non-diagonal elements and store it in a vector such that the $i$-th coordinate of that vector is the sum $R_i$ of the elements of the $i$-th row of the matrix.

   *Note.* An easy way to compute the sum of the absolute values of the elements of all the rows of a matrix `M` in `Python` is to use the following code:

```
rows_sum = np.sum(np.abs(M), axis=1)  # axis=1 means sum over columns
```

Recall that so far the values in the diagonal are zero (by construction). So the sum computed above is exactly $R_i$ for every row $i$.

5. Finally, fill the diagonal elements of the matrix such that $|a_{ii}| > R_i$ for every row $i$. A simple way of doing this is to set $a_{ii} = R_i + \varepsilon$, where $\varepsilon$ is a random number in the interval $(\delta, 1)$, with $\delta = 0.01$ (just in case).
6. Finally, return the matrix $\mathbf{A} = \mathbf{M} + \mathbf{D}$.