TP 2 : Décomposition LU.

On cherche à résoudre le système $Ax = b$ où $A \in \mathcal{M}_n(\mathbb{R})$ et $b \in \mathbb{R}^n$.

1) Écrire une fonction `[x]=solveTinf(L,b)` permettant de résoudre le système $Lx = b$ dans le cas où la matrice L est triangulaire inférieure. On vérifiera son bon fonctionnement sur des matrices de petite taille. On rappelle que la solution de ce système vaut

$$x_i = \left(b_i - \sum_{k=1}^{i-1} L_{i,k} x_k \right) / L_{i,i}.$$

1bis) Écrire une fonction `[x]=solveTsup(U;b)` permettant de résoudre le système $Ux = b$ dans le cas où U est triangulaire supérieure. On vérifiera son bon fonctionnement sur des matrices de petite taille. On rappelle que la solution de ce système vaut

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

2) Écrire une fonction `[L,U]=factorLU(A)` qui prend en entrée une matrice carrée A et qui renvoie les matrices L, U de sa factorisation LU. On rappelle le pseudo code de cette décomposition [1] :

pour $i = 1, \dots, n$ **faire**
  pour $k = i, \dots, n$ **faire**
    $U_{ik} \leftarrow A_{ik} - \sum_{j=1}^{i-1} L_{ij} U_{jk}$
  fin pour
  $L_{ii} \leftarrow 1$
  pour $k = i+1, \dots, n$ **faire**
    $L_{ki} \leftarrow \frac{1}{U_{ii}} \left(A_{ki} - \sum_{j=1}^{i-1} L_{kj} U_{ji} \right)$
  fin pour
fin pour

On fera en sorte que la fonction renvoie un message d'erreur si un des pivots U_{ii} est nul [2]. Ici aussi, on vérifiera que la fonction marche bien sur des exemples.

3) Écrire une fonction `[x]=solveLU(A,b)` permettant de résoudre le système $Ax = b$ en passant par la factorisation $A = LU$. La fonction devra s'écrire en trois instructions. Vérifier que cette fonction marche bien sur des exemples (on pourra comparer avec la solution donnée par `A\b`, ou bien mesurer la norme de $Ax - b$).

1. Une somme réalisée sur un ensemble d'indices vide est égale à zéro.
2. Pour afficher un message d'erreur, et arrêter en même temps l'exécution de la fonction en cours, on utilisera la fonction `error`.

4)

1. Écrire une fonction `[A]=Laplacien(n)` qui prend en entrée un entier n, et renvoie une matrice du Laplacien en dimension n, définie par

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \cdots & 0 & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix}, \qquad h = \frac{1}{n+1}.$$

   On pourra réutiliser le code d'une fonction du TP1.

2. Considérons le système $Ax = b$ lorsque A est une matrice du Laplacien, et b est un vecteur dont tous les coefficients sont 1. Utiliser la fonction `solveLU` pour le résoudre, et mesurer le temps de calcul pour trouver la solution. On utilisera pour cela la fonction `timer` (revoir le TP1). Comparer avec le temps pris par la fonction `A\b`, qui est elle aussi codée via une décomposition LU. Que constate-t-on ?
3. En gardant le contexte de la question précédente, écrire une fonction `plotLUtesterror` qui prend en entrée un entier n, et qui retourne deux quantités : le temps de calcul de la fonction `solveLU`, et l'écart avec la solution du système `A\b` (en norme $\|\cdot\|_\infty$).
4. Utiliser la fonction `plotLUtesterror` pour tracer le temps de calcul de la fonction `solveLU` en fonction de la taille de la matrice (On pourra faire varier n entre 1 et 150).
5. Vérifier expérimentalement que le temps de calcul de la méthode est de l'ordre de $\mathcal{O}(n^3)$. Noter que c'est équivalent à vérifier que la racine cubique du temps évolue en $\mathcal{O}(n)$.

5) Résoudre le système lorsque $A = \begin{pmatrix} 10^{-17} & 1 \\ 1 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$. Que constate-t-on ? On pourra également remplacer A_{11} par 10^{-n} pour d'autres valeurs de n.

Remarque : le phénomène observé s'explique à l'aide de la notion de *conditionnement* de la matrice A, qui sera vue en cours dans le chapitre 2.