Résolution de systèmes linéaires

L'objectif de ce TP est de programmer des méthodes numériques de résolution de système linéaire $Ax = b$ et de calcul de l'inverse A^{-1}, et d'étudier la complexité de ces méthodes.
On placera les fichiers dans un nouveau répertoire `TP_syslin` (systèmes linéaires).

1 Produit de matrices

1) Écrire une fonction `C=produit(A,B)` qui prend en argument deux matrices `A` et `B` et renvoie leur produit, ou affiche un message d'erreur si les dimensions sont incompatibles. On utilisera la fonction `error("message d'erreur")` pour afficher l'erreur et interrompre la fonction. On pourra utiliser `[m,n]=size(A)` pour obtenir la taille de `A`. On utilisera des boucles et pas d'opération sur les matrices.

2) Définir des matrices `A`, `B`, `C` de tailles (5,4),(4,7) et (7,6) respectivement, remplies de valeurs réelles aléatoires dans [0,1] (utiliser `rand(m,n)` qui renvoie une telle matrice, de taille (m,n)), et tester l'égalité des produits $(AB)C$ et $A(BC)$.

3) Modifier la fonction en `[C,nb_a,nb_m]=produit(A,B)` pour renvoyer aussi `nb_a` et `nb_m` qui sont le nombre d'additions et de multiplications effectuées. Puis, dans un script `complex_produit.m`, représenter graphiquement ces nombres lorsque les matrices `A` et `B` sont carrées de taille (n,n) pour $n = 1,...,100$.

2 Résolution de systèmes triangulaires

On commence par des systèmes ayant des formes particulières.
Soit A une matrice carrée de taille (n,n) et b un vecteur de taille $(n,1)$ (vecteur colonne).
Si A est diagonale, alors le système $Ax = b$ se résout directement : les équations sont $a_{i,i}x_i = b_i$ pour $i = 1,\dots,n$ donc $x_i = \cdots$

4) Écrire une fonction `x=sys_diag(A,b)` qui prend en argument une matrice carrée `A` **diagonale** et un vecteur colonne `b` (ayant autant de lignes que `A`), et renvoie la solutioni `x` de $Ax = b$ (on supposera les coefficients diagonaux de A non nuls).

5) Proposer un test pour cette fonction avec une matrice et un vecteur de taille 3 particuliers.

Si A est triangulaire inférieure, le système $Ax = b$ se résout ligne à ligne à partir de la première équation (qui fournit x_1) puis en substituant les valeurs trouvées dans l'équation suivante.

6) Que vaut x_1 ? Écrire (sur papier) l'expression du coefficient $(Ax)_i$ puis exprimer x_i (où $Ax = b$) en fonction de $x_1,\dots,x_{i-1}$ lorsque $i \geq 2$.

7) Écrire une fonction `x=sys_tri_inf(A,b)` qui prend en argument une matrice carrée `A` **triangulaire inférieure** et un vecteur colonne `b` (ayant autant de lignes que `A`), et renvoie la solution `x` de $Ax = b$ (on supposera les coefficients diagonaux de A non nuls).

8) Proposer un test pour cette fonction avec une matrice et un vecteur de taille 10 choisis aléatoirement.

Si A est triangulaire supérieure, le système $Ax = b$ se résout ligne à ligne à partir de la dernière équation (qui fournit x_n) puis en substituant les valeurs trouvées dans l'équation précédente.

9) Que vaut x_n ? Écrire (sur papier) l'expression du coefficient $(Ax)_i$ puis exprimer x_i (où $Ax = b$) en fonction de $x_{i+1}, \ldots, x_n$ lorsque $i < n$.

10) Écrire une fonction `x=sys_tri_sup(A,b)` qui prend en argument une matrice carrée `A` **triangulaire supérieure** et un vecteur colonne `b` (ayant autant de lignes que `A`), et renvoie la solution `x` de $Ax = b$ (on supposera les coefficients diagonaux de A non nuls).

11) Proposer un test pour cette fonction avec une matrice et un vecteur de taille 10 choisis aléatoirement.

3 Matrices quelconques, pivot de Gauss

La méthode du pivot de Gauss consiste à ramener le système $Ax = b$ à un système triangulaire supérieur, à l'aide d'opérations sur les lignes (effectuées sur A et sur b).
Tout d'abord, les opérations $L_i \leftarrow L_i - \frac{a_{i,1}}{a_{1,1}} L_1$ (pour $i = 2, \ldots, n$) ramènent à une première colonne ayant uniquement son premier coefficient non nul.
Puis, sur la matrice A' obtenue, les opérations $L_i \leftarrow L_i - \frac{a'_{i,2}}{a'_{2,2}} L_1$ (pour $i = 3, \ldots, n$) ramènent à une deuxième colonne ayant uniquement ses deux premiers coefficients non nuls.
Et de même pour toutes les colonnes suivantes.
NB. On suppose ici que les "pivots" $a_{1,1}$, $a'_{2,2}$, etc., sont non nuls, ce qui permet de faire ces opérations.

12) Écrire une fonction `x=sys_gauss(A,b)` qui prend en argument une matrice carrée `A` et un vecteur colonne `b` (ayant autant de lignes que `A`), et renvoie la solution `x` de $Ax = b$ (on supposera les coefficients diagonaux de A non nuls). Pour cela, on effectuera les opérations sur A et b pour se ramener au cas triangulaire supérieur, puis on appellera la fonction `sys_tri_sup` précédente pour finalement résoudre le système. NB. Il est possible de modifier `A` et `b` dans la fonction, cela n'affectera pas la valeur de ces variables en-dehors de la fonction (Octave crée des "copies" de ces variables qui existent au sein de la fonction).

13) Proposer un test pour cette fonction avec une matrice et un vecteur de taille 10 choisis aléatoirement.

14) Modifier la fonction en `[x,nb_a,nb_m]=sys_gauss(A,b)` pour renvoyer aussi `nb_a` et `nb_m` qui sont le nombre d'additions et de multiplications effectuées. Puis, dans un script `complex_gauss.m`, représenter graphiquement ces nombres lorsque les matrices `A` et `B` sont carrées de taille (n,n) pour $n = 1,...,100$.

Il reste à prendre le compte le cas où un pivot serait nul. Plutôt que de tester la nullité (à une marge d'erreur près...), il vaut mieux, de façon systématique, lors du travail sur la colonne j, choisir le plus grand pivot $a_{i,j}$ (avec $i \geq j$) de la colonne (en valeur absolue), et on *échange* alors les lignes i et j de A (et donc de b).

15) Écrire une fonction `x=sys_gauss_maxpivot(A,b)` qui résout le système linéaire en prenant en compte ces échanges de lignes pour maximiser les pivots.

16) Écrire une fonction `d=determinant(A)` qui calcule le déterminant de A. Pour cela, on reprendra le code de la fonction précédente. Pour rappel, le déterminant ne change pas lors d'une opération $L_i \leftarrow L_i + \lambda L_j$ (où $j \neq i$), est multiplié par -1 lors d'un échange de lignes, et vaut le produit des termes diagonaux pour une matrice triangulaire. Adapter le code pour renvoyer 0 lorsque l'algorithme du pivot échoue (i.e. si un pivot est nul malgré la recherche d'échange), ce qui implique que la matrice est singulière ; on pourra utiliser `return` pour quitter immédiatement une fonction (sans déclencher d'erreur).

4 Inversion de matrice

Inverser une matrice A revient à calculer les solutions $x_1,\ldots,x_n$ de $Ax_1 = e_1,\ldots, Ax_n = e_n$, où $e_1,\ldots,e_n$ est la base canonique de $\mathbb{R}^n$, car on a alors

$$A^{-1} = \left(\begin{array}{c|c|c} x_1 & \cdots & x_n \end{array} \right).$$

Il s'agit donc de résoudre n systèmes. Néanmoins, une grande part du calcul (réduction de A à une forme triangulaire) est commune à tous ces systèmes ; il faut par contre effectuer les opérations sur les n seconds membres, et *in fine* résoudre n systèmes triangulaires.

17) Écrire une fonction `X=matrice_inverse(A)` qui prend en argument une matrice carrée inversible et renvoie son inverse. On adaptera `sys_gauss_maxpivot` pour tenir compte de n seconds membres $e_1,\ldots,e_n$ stockés dans une matrice B (initialement, $B = I_n$). *Attention* (notamment lors de l'échange de lignes), les premières colonnes de B n'ont pas la même structure triangulaire que A donc les indices à considérer peuvent différer de ceux de A.