Feuille de TP N°03 – Systèmes non Triangulaires (Solution)

Exercice 01 : Calcul du déterminant

1. Ecrire une fonction Matlab récursive « determinant » qui prend comme argument une matrice carrée quelconque *A* et qui retourne comme résultat le déterminant de cette dernière.
2. Tester cette fonction sur une matrice de votre choix, et comparer le résultat à celui de la fonction Matlab prédéfinie « det » pour vérifier que votre résultat est juste.

Rappel de cours : le déterminant d'une matrice carrée *A* d'ordre *n* peut être calculé avec la formule suivante (en choisissant une ligne *i*) :

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} * a_{i,j} * \det(A_{i,j})$$

Avec, $a_{i,j}$ est l'élément à la ligne *i* et la colonne *j* de la matrice *A*, et $A_{i,j}$ est la matrice *A* sans la ligne *i* et la colonne *j*.

Pour une matrice carrée *A* d'ordre 2 nous avons :

$$\det(A) = a_{1,1} * a_{2,2} - a_{2,1} * a_{1,2}$$

```
function d = determinant(A)
    [m n] = size(A);
    if m == n
        if n == 1
            d = A(1, 1);
        elseif n == 2
            d = A(1, 1) * A(2, 2) - A(1, 2) * A(2, 1);
        else
            d = 0;
            for j = 1:n
                d = d + (-1)^(1 + j) * A(1, j) *
                            determinant(A(2:end, [1:j-1 j+1:n]));
            end
        end
    else
        d = 0;
        display('La matrice n''est pas carree');
    end
end
```

Exercice 02 : Elimination de Gauss

1. Ecrire une fonction « elimination_gauss » qui prend comme argument une matrice de coefficients quelconque A et un vecteur de second membre b, et qui applique l'élimination de Gauss pour retourner une matrice triangulaire supérieure A_2 et un vecteur colonne b_2 tel que :
    a. Le système $A_2x = b_2$ est équivalent au système du départ $Ax = b$ (les deux systèmes ont la même solution).
    b. La matrice de coefficients retournée A_2 doit être triangulaire supérieure.

```
function [A2 b2] = elimination_gauss(A, b)
    [m n] = size(A);
    if m == n
        A2 = A;
        b2 = b;
        for i = 1:n
            for j = i+1:n
                c = A2(j, i) / A2(i, i);
                b2(j) = b2(j) - b2(i) * c;

                % Solution 01 – Boucle
                for k = i:n
                    A2(j, k) = A2(j, k) - A2(i, k) * c;
                end

                % Solution 02 – Opération Vectorielle
                % A2(j, i:n) = A2(j, i:n) - A2(i, i:n) .* c;
            end
        end
    else
        display('La matrice de coefficients n''est pas carree');
    end
end
```

2. Tester la fonction d'élimination de gauss sur un système linéaire aléatoire de taille 100.
    a. Utiliser la fonction « type_matrice » pour vérifier que la matrice A_2 du résultat est triangulaire supérieure.
    b. Qu'observez-vous ? expliquer.

    Dans ce cas, on observe que la matrice obtenue n'est pas triangulaire. Mathématiquement l'opération « a – b * a / b » utilisée pour éliminer les éléments en dessous de la diagonale doit toujours donner zéro. Néanmoins, dans le cadre de l'arithmétique flottante, des erreurs d'arrondi sont introduite et des résultats proches de zéro peuvent être obtenus. Par conséquent, si nous testons si la matrice est triangulaire, le programme va trouver que non, car la matrice contient des valeurs non nulles en dessous de la diagonale principale.

    Le code suivant résout ce problème :

```
function [A2 b2] = elimination_gauss(A, b)
```

```
    [m n] = size(A);
    if m == n
        A2 = A;
        b2 = b;
        for i = 1:n
            for j = i+1:n
                c = A2(j, i) / A2(i, i);
                b2(j) = b2(j) - b2(i) * c;
                A2(j, i) = A2(j, i) - A2(i, i) * c;

                % Solution 01 – Boucle
                for k = i+1:n
                    A2(j, k) = A2(j, k) - A2(i, k) * c;
                end

                % Solution 02 – Opération Vectorielle
                % A2(j, i+1:n) = A2(j, i+1:n) - A2(i, i+1:n) .* c;
            end
        end
    else
        display('La matrice de coefficients n''est pas carree');
    end
end
```

3. Utiliser votre fonction pour transformer un système linéaire aléatoire de taille 100 en un système triangulaire équivalent, puis résoudre le système obtenu avec la fonction « resoudre_triangulaire_superieur » que vous avez implémentée dans la deuxième feuille de TP.
    a. Vérifier que $b - Ax = 0$ (vecteur nul) pour confirmer que votre solution est juste.
    b. Qu'observez-vous ? expliquer.

    Dans ce cas, est pour les mêmes raisons liées à l'arithmétique flottante expliquées dans l'exercice 01, on observe que « b – Ax ≠ 0 ». A cause des erreurs d'arrondi, la résolution du système linéaire « Ax = b » ne va pas fournir un résultat exact « x » mais plutôt une approximation de cette dernière. Le résultat de l'opération « b – Ax » est appelé « l'erreur inverse ». Il dépend de la taille du système et de son conditionnement.

    c. Comparer vos résultats à ceux obtenus par le solveur prédéfini de Matlab ($x = A\backslash b$) pour confirmer d'une autre façon que votre algorithme est juste.

    Les deux solutions (celle calculé par notre fonction et celle prédéfinies dans Matlab) dans des solutions proches numériquement, mais différentes. Dans les deux cas, « b – Ax ≠ 0 ». Cela est dû aux algorithmes effectuant les opérations dans un ordre différent, et par conséquent les erreurs flottantes commises sont différente entre les deux fonctions.

Exercice 03 : Factorisation LU

1. Ecrire une fonction Matlab « factorisation_lu » qui prend comme argument une matrice quelconque *A* et qui retourne comme résultat une matrice triangulaire inférieure *L* et une matrice triangulaire supérieure *U* tel que : *A = LU*.

```
function [L U] = factorisation_lu(A)
    [m n] = size(A);
    if m == n
        L = eye(n);
        U = A;
        for i = 1:n
            for j = i+1:n
                L(j, i) = U(j, i) / U(i, i);
                U(j, i) = 0;

                % Solution 01 - Boucle
                for k = i+1:n
                    U(j, k) = U(j, k) - U(i, k) * L(j, i);
                end

                % Solution 02 – Opération Vectorielle
                % U(j, i+1:n) = U(j, i+1:n) - U(i, i+1:n).*L(j, i);

            end
        end
    else
        display('La matrice n''est pas carree');
    end
end
```

2. Ecrire une fonction « resoudre_lu » qui résout un système linéaire en utilisant la factorisation LU de la matrice de coefficients *A.* Puis tester cette fonction et comparer ces résultats à ceux obtenus avec le solveur prédéfini dans Matlab pour vérifier que votre fonction est juste.

```
function x = resoudre_lu(A, b)
    [L U] = factorisation_lu(A);
    y = resoudre_triangulaire_inferieure(L, b);
    x = resoudre_triangulaire_superieure(U, y);
end
```

3. Ecrire une fonction « determinant_lu » qui prend comme argument une matrice quelconque *A* et qui calcule son déterminant à l'aide de la factorisation *LU*.
    - Le calcul est basé sur la propriété « *det(A) = det(LU) = det(L) * det(U)* ».
    - Utiliser la fonction « determinant_triangulaire » de la feuille de TP 02 pour calculer les déterminants des matrices *L* et *U*.

```
function d = determinant_lu(A)
    [L U] = factorisation_lu(A);
    d = determinant_triangulaire(L) * determinant_triangulaire(U);
end
```

4. Comparer les résultats et le temps d'exécution de la fonction « determinant_lu » à ceux de la fonction « determinant » (de l'exercice 01 de cette feuille de TP) en effectuant le calcul sur une matrice aléatoire carrée d'ordre 10.
    - Qu'observez-vous ? Expliquer.

Le calcul du déterminant en utilisant la factorisation LU est considérablement plus rapide par rapport au calcul basé sur la décomposition matricielle. Cela est dû à la complexité algorithmique trop élevée du premier algorithme qui est de O(n!) par rapport à la factorisation LU qui est de $O(n^2)$.

Exercice 04 : Elimination de Gauss avec Pivot Partiel

1. Ecrire une fonction « gauss_pivot_partiel » qui prend comme argument une matrice de coefficients quelconque *A* et un vecteur de second membre *b*, et qui applique l'élimination de Gauss avec pivot partiel pour éviter les pivots nuls sur des matrices ayant un déterminant non nul.
    a. Le pivot doit être déterminé en choisissant la valeur la plus large en valeur absolue en dessous de la diagonale sur la même colonne.
    b. La fonction doit dire si la matrice de coefficients n'est pas inversible.

```
function [A2 b2] = gauss_pivot_partiel(A, b)
    [m n] = size(A);
    if m == n
        A2 = A;
        b2 = b;
        for i = 1:n
            % Solution 01 - Boucles
            imax = i;
            for index = i+1:n
                if abs(A2(index, i)) > abs(A2(imax, i))
                    imax = index;
                end
            end
            if A2(imax, i) == 0
                display('La matrice A n''est pas inversible');
            end
            t = b2(imax);
            b2(imax) = b2(i);
            b2(i) = t;
            for k = 1:n
                t = A2(imax, k);
                A2(imax, k) = A2(i, k);
                A2(i, k) = t;
            end

            % Solution 02 – Opérations Vectorielles
            % [m, imax] = max(abs(A2(i:n, i)));
            % imax = imax + i - 1;
```

```
            % if A2(imax, i) == 0
            %     display('la matrice A n''est pas inversible');
            % end
            % if (imax ~= i)
            %     A2([imax i], :) = A2([i imax], :);
            %     b2([imax i]) = b2([i imax]);
            % end

            % Fin de la recherché de pivot.
            % Nous procédons à l'élimination de Gauss.
            for j = i+1:n
                c = A2(j, i) / A2(i, i);
                b2(j) = b2(j) - b2(i) * c;
                A2(j, i) = 0;
                A2(j, i+1:n) = A2(j, i+1:n) - A2(i, i+1:n) .* c;
            end
        end
    else
        display('La matrice de coefficients n''est pas carree');
    end
end
```

2. Soit le système linéaire déterminé par la matrice de coefficients et le vecteur de second membre suivants :

$$A = \begin{pmatrix} -1 & -1 & 3 & 9 & -4 \\ 2 & 2 & 2 & -3 & 0 \\ 4 & 4 & -6 & 1 & 3 \\ -6 & 5 & -7 & -5 & 7 \\ 0 & -4 & 0 & 5 & 9 \end{pmatrix}, b = \begin{pmatrix} 36 \\ 5 \\ 33 \\ -53 \\ -55 \end{pmatrix}.$$

   a. Essayer de résoudre ce système en utilisant la méthode de gauss sans échange (de l'exercice 02) puis en utilisant l'élimination de Gauss avec pivot partiel.

L'élimination de Gauss sans échange ne donne pas de solution valide, elle fournit des valeurs « NaN » et « Inf » à cause d'une division par zéro sur la deuxième colonne. Ceci arrive malgré le fait que la matrice A est inversible, la méthode de Gauss avec échange de pivot permet de résoudre ce problème en utilisant le plus grand pivot sur la colonne de l'élément à éliminer (en dessous de la diagonale). Dans ce cas, ces erreurs apparaissent si et seulement si la matrice de coefficients n'est pas inversible (a un déterminant nul).

   b. Qu'observez-vous ? expliquer.

3. Initialiser un système linéaire aléatoire de taille 15, puis résoudre ce même système dans un premier temps en utilisant la méthode de Gauss sans échange (garder la solution dans une variable x_1), puis avec pivot partiel (garder cette solution dans une variable x_2).
   a. Comparer les deux solutions et dire laquelle est la plus précise.

b. ***Indication :*** une solution est plus précise qu'une autre si elle assure un résidu ($b - Ax$) inférieur.

Dans ce cas, nous observons que la solution calculée en utilisant l'élimination de Gauss sans échange est moins précise que la solution calculée avec l'élimination de Gauss avec échange. On dit que l'algorithme d'élimination de Gauss avec échange est numériquement plus stable