1 Résolution de systèmes d'équations linéaires

1.1 Méthode de Gauss-Jordan

La méthode de Gauss-Jordan est une technique de résolution de systèmes d'équations linéaires :

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix} \tag{1}$$

Elle consiste à transformer la matrice associée $A = (a_{ij})_{0 \le i < n,\, 0 \le j < n}$ en une matrice unité en effectuant des combinaisons linéaires de lignes. Cette variante de la méthode de Gauss permet de calculer l'inverse d'une matrice.

1.2 Matrice augmentée

Soient A et B deux matrices ayant le même nombre de lignes, on appelle matrice augmentée la matrice $(A|B)$ formée des deux blocs A et B mis côte à côte.

Cette notion est très pratique lorsqu'on veut résoudre le système d'équations avec différents seconds membres. La matrice B est alors construite en accollant à droite de A, les vecteurs colonnes formant les seconds membres. Elle se réduit naturellement à un vecteur dans le cas d'un seul second membre.

1.3 Algorithme de base sans changement de pivot

On augmente la matrice A avec le(s) vecteur(s) colonne(s) du second membre. On vérifie que $\forall i, a_{ii} \neq 0$ sinon effectue des échanges de lignes.

L'algorithme est itératif et s'effectue en n étapes. A l'étape $k \in [0, n[$, on combine toutes les lignes sauf la ligne k pour faire apparaître des 0 sur toute la colonne k sauf au niveau du pivot a_{kk}.

pour k=0 à n-1
    diviser la ligne k de la matrice A par a_{kk}
    pour i=0 à n-1 sauf k
        retrancher à la ligne i, la nouvelle ligne k multipliée par a_{ik}
    fin pour i
fin pour k

Après résolution, la matrice A apparait comme la concaténation de la matrice identité, accolée aux solutions x du système.

1.4 Exemple pratique

Soit à résoudre :

$$\begin{cases} 2x + y - 4z &= 8 \\ 3x + 3y - 5z &= 14 \\ 4x + 5y - 2z &= 16 \end{cases}$$

La matrice augmentée $A^{(0)}$ s'écrit :

$$A^{(0)} = \left(\begin{array}{ccc|c} 2 & 1 & -4 & 8 \\ 3 & 3 & -5 & 14 \\ 4 & 5 & -2 & 16 \end{array}\right) \tag{2}$$

Divisons la première ligne par le pivot $a_{0,0}$: $l_0 \leftarrow \frac{1}{2} \times l_0$

$$A^{(1)} = \left(\begin{array}{ccc|c} 1 & 1/2 & -2 & 4 \\ 0 & 3/2 & 1 & 2 \\ 0 & 3 & 6 & 0 \end{array}\right) \begin{array}{l} \\ l_1 \leftarrow l_1 - 3 \times l_0 \\ l_2 \leftarrow l_2 - 4 \times l_0 \end{array} \tag{3}$$

Divisons la deuxième ligne par le pivot a_{11} : $l_1 \leftarrow \frac{2}{3} \times l_1$

$$A^{(2)} = \left(\begin{array}{ccc|c} 1 & 0 & -7/3 & 10/3 \\ 0 & 1 & 2/3 & 4/3 \\ 0 & 0 & 4 & -4 \end{array}\right) \begin{array}{l} l_0 \leftarrow l_0 - \frac{1}{2} \times l_1 \\ \\ l_2 \leftarrow l_2 - 3 \times l_1 \end{array} \tag{4}$$

Divisons la troisième ligne par le pivot $a_{2,2}$: $l_2 \leftarrow \frac{1}{4} \times l_2$

$$A^{(3)} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -1 \end{array}\right) \begin{array}{l} l_0 \leftarrow l_0 + \frac{7}{3} \times l_2 \\ l_1 \leftarrow l_1 - \frac{2}{3} \times l_2 \\ \\ \end{array} \tag{5}$$

La solution se trouve dans la quatrième colonne. Vérifier que $x = (1, 2, -1)^t$ est bien solution.

1.5 Inverse d'une matrice

Le calcul direct de l'inverse d'une matrice A mettant en oeuvre la formule $A^{-1} = \frac{1}{det(A)} {}^t co(A)$ est peu utilisé en pratique. On préfère résoudre n systèmes linéaires en même temps en plaçant dans le second membre la matrice identité I_n. On applique la méthode de Gauss-Jordan à la matrice augmentée $(A|I_n)$.

- Montrer que l'inverse de la matrice
$A = \begin{pmatrix} 2 & 1 & -4 \\ 3 & 3 & -5 \\ 4 & 5 & -2 \end{pmatrix}$ est $A^{-1} = \frac{1}{12} \begin{pmatrix} 19 & -18 & 7 \\ -14 & 12 & -2 \\ 3 & -6 & 3 \end{pmatrix}$. Vérifier que $AA^{-1} = I$.

1.6 Implémentation de Gauss-Jordan en Python

1. Proposez une implémentation d'une fonction `gauss_jordan` en python de l'algorithme précédent, permettant de résoudre un système d'équations linéaires. On limitera la précision d'affichage avec l'instruction : `numpy.set_printoptions(precision=4, suppress=True)`. L'option `suppress=True` supprime les petits zéros lors de l'affichage.
2. Traitez quelques cas tests, vérifier vos résultats obtenus avec la fonction `numpy.linalg.solve`
3. On étudie ici la stabilité numérique de l'algorithme en prenant :
$A = \begin{pmatrix} 1 & 1/4 & 0 & 0 \\ 1 & 5/4 & 12 & 0 \\ 1 & 1/3 & 1 & 1 \\ 1 & 5/4 & 13 & 1 \end{pmatrix}$ et $B = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$
Calculer le résidu $B - A\,x$ dans ce cas, commenter.
4. On émettra une erreur si le pivot $|a_{k,k}| < \epsilon \approx 10^{-6}$. Pour ce faire on utilisera l'instruction `raise` dans la fonction`gauss_jordan` pour lancer une erreur de type `ZeroDivisionError`.

```
if abs(akk)<epsilon:
    raise ZeroDivisionError("pivot akk trop faible")
```

On récupérera l'erreur de type `ZeroDivisionError` dans le programme appelant en utilisant la structure :

```
try:
    gauss_jordan(K, L)
except ZeroDivisionError as err:
    print('Handling run-time error:', err)
```

1.7 Algorithme de Gauss-Jordan avec changement de pivot

On augmente la matrice A avec le(s) vecteur(s) colonne(s) du second membre. On propose une nouvelle version de l'algorithme de Gauss-Jordan avec maximisation du pivot. L'algorithme est itératif et s'effectue en n étapes. A l'étape $k \in [0, n[$, on combine toutes les lignes sauf la ligne k pour faire apparaître des 0 sur toute la colonne k sauf au niveau du pivot a_{kk}.

pour k=0 à n-1
    identifier l'indice de ligne $i \in [k, n[$ correspondant au $\max\{|a_{i,k}|\}$
    échanger les lignes i et k de la matrice A
    diviser ligne k de la nouvelle matrice A par a_{kk}
    pour i=0 à n-1 sauf k
        retrancher à la ligne i, la nouvelle ligne k multipliée par a_{ik}
    fin pour i
fin pour k

1. traiter quelques cas tests.
2. refaites l'étude de la stabilité numérique.

1.8 Résolution de systèmes mal conditionnés

On examine ici l'influence d'une perturbation δb sur les termes du second membre b du système d'équations linéaires. Il en résulte une variation δx sur la solution. On montre par contre que δx n'est pas forcément une perturbation de x et dépend du conditionnement de A. On a l'inégalité suivante :

$$\frac{\|\delta x\|}{\|x\|} < cond(A)\frac{\|\delta b\|}{\|b\|} \tag{6}$$

.

Le conditionnement est donné dans le cas d'une matrice symétrique par le rapport des valeurs propres extrêmes en norme de A : $\kappa(A) = \frac{|\lambda_{max}(A)|}{|\lambda_{min}(A)|}$.

1. Résoudre le système d'équations en prenant : $A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$ et $B = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$
2. Faites une fluctuation de ±1% sur le second membre et en déduire l'erreur sur la solution.
3. Calculer le conditionnement de A à partir de son spectre. On utilisera la fonction `numpy.linalg.eig` pour calculer les valeurs propres.
4. Pour vérifier l'inégalité (**??**), on calculera les normes avec la fonction `numpy.linalg.norm` en choisissant une norme infinie, par exemple.

2 Décomposition LU d'une matrice avec pivot

Le but est ici de décomposer une matrice sous la forme : $A = P\,L\,U$ où P est une matrice de permutation de lignes de la matrice unité, L une matrice triangulaire inférieure ne contenant que des 1 sur sa diagonale et U une matrice triangulaire supérieure. La matrice de permutation est une matrice orthogonale et vérifie la propriété $P^{-1} = P^t$.

A titre d'exemple, factorisons la matrice $A = \begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix}$

Déterminons le pivot qui correspond à l'amplitude maximale des termes de la première sous-colonne. La première étape de permutation est triviale parce que la valeur 10 est déjà la plus grande valeur. la première matrice de permutation est l'identité.

$$P_0\, A^{(0)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} = \begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \tag{7}$$

La première étape d'élimination consiste à faire les combinaisons linéaires suivantes – Numérotation des lignes conformes à python – :

$$\begin{aligned} l_0 &\leftarrow l_0 \\ l_1 &\leftarrow l_1 + 3/10 \times l_0 \\ l_2 &\leftarrow l_2 - 1/2 \times l_0 \end{aligned} \tag{8}$$

$$A^{(1)} = L_0^{-1}\, P_0\, A^{(0)} = \begin{pmatrix} 1 & 0 & 0 \\ 3/10 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & -1/10 & 6 \\ 0 & 5/2 & 5 \end{pmatrix} \tag{9}$$

La seconde étape de permutation revient à échanger les lignes 1 et 2 :

$$P_1\, A^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ 0 & -1/10 & 6 \\ 0 & 5/2 & 5 \end{pmatrix} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 5/2 & 5 \\ 0 & -1/10 & 6 \end{pmatrix} \tag{10}$$

La seconde étape d'élimination consiste à faire l'opération suivante :

$$l_2 \leftarrow l_2 + \frac{1}{25} \times l_1 \tag{11}$$

$$U = A^{(2)} = L_1^{-1}\, P_1\, A^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/25 & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ 0 & 5/2 & 5 \\ 0 & -1/10 & 6 \end{pmatrix} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 5/2 & 5 \\ 0 & 0 & 31/5 \end{pmatrix} \tag{12}$$

On construit finalement la matrice globale de permutation $P = P_0\, P_1$.
Le calcul des matrices L_k s'effectue facilement grâce à la propriété suivante :

$$L_k = L_k^{-1} + 2 I_n \tag{13}$$

La matrice triangulaire inférieure L est finalement reconstruite en appliquant la relation suivante :

$$L = P^t\, \Pi_{k=0}^{n-2} (P_k\, L_k) \tag{14}$$

Dans l'algorithme fourni ci-après, on définit une fonction `Find_Pivot(A, k)` qui retourne pour une colonne k donné, l'indice de ligne $i \in [k, n[$ correspondant au $\max\{|a_{i,k}|\}$.

```
function LU(A)
    PL ← I_n
    P ← I_n
    for k ∈ [0, n−1[ do
        P_k ← I_n
        i ← Find_Pivot(A, k)
        P_k ← Swap_Lines(P_k, i, k)
        A ← Swap_Lines(A, i, k)

        invL_k ← I_n
        for i ∈ [k+1, n−1[ do
            invL_k(i, k) ← −a_{i,k}/a_{k,k}
        end for

        A ← Mult(invL_k, A)
        L_k ← −invL_k + 2I_n
        PL_k ← Mult(P_k, L_k)
        PL ← Mult(PL, PL_k)
        P ← Mult(P, P_k)
    end for

    U ← A
    invP ← Transpose(P)
    L ← Mult(invP, PL)
    return P, L, U
end function
```

1. Proposez une implémentation en python.
2. Testez-la en utilisant les matrices fournies précédemment.