Systèmes Linéaires : Pivot de Gauss

1 Pivot de Gauss

1.1 Exemple ; transvections

Exercice 1 : Résoudre le système d'équations linéaires suivants :

$$(S) : \begin{cases} 2x & + & 2y & - & 3z & = & 2 \\ -2x & - & y & - & 3z & = & -5 \\ 6x & + & 4y & + & 4z & = & 16 \end{cases}$$

Exercice 2 : Écrire le système précédent sous forme matricielle

On appelle transvection une opération élémentaire du type : "on ajoute tant de fois telle équation à telle autre". Nous n'effectuerons des transvections que sur les lignes et on écrira donc :

$$L_i \leftarrow L_i + \mu L_j$$

On suppose que le coefficient en x en première équation est non nul (lorsque ça ne sera pas le cas on échangera des lignes). On va s'en servir comme **pivot** pour éliminer les autres occurrences de x.

On rappelle qu'un système est de Cramer lorsqu'il admet une unique solution. Lorsque c'est le cas, il n'y aura pas de problème de disparition du pivot dans une équation, et nous supposerons qu'il en sera toujours ainsi dans le cadre de ce cours.

1.2 Problème de comparaison à zéro

Exercice 3 : Commencer à résoudre le système suivant jusqu'à rencontrer un problème :

$$(S) : \begin{cases} 10^{-17}x & + & y & = & 1 \\ x & + & y & = & 2 \end{cases}$$

Pour limiter les mauvaises surprises, on va prendre comme pivot le coefficient le plus élevé en valeur absolue. Cela s'appelle la méthode du *pivot partiel*.

1.3 Formalisation de l'algorithme

On cherche à résoudre AX=Y avec A=$(a_{i,j})_{1\leqslant i,j\leqslant n}$ inversible.

Étape 1 : Mise sous forme triangulaire

pour i de 0 à (n-2) faire
    Trouver k entre i et n-1 tel que $|a_{k,i}|$ soit maximal
    Échanger L_k et L_i
    Pour m variant de i+1 à n-1 faire
        $L_m \leftarrow L_m - \dfrac{a_{m,i}}{a_{i,i}} L_i$

Étape 2 : Phase de remontée

On a $x_i = \dfrac{1}{a_{i,i}} \left(y_i - \displaystyle\sum_{k=i+1}^{k=n-1} a_{i,k} x_k \right)$

pour i de (n-1) à 0 faire
    pour k de i+1 à n-1 faire
        $y_i \leftarrow y_i - a_{i,k} x_k$
    $y_i \leftarrow \dfrac{y_i}{a_{i,i}}$

2 Mise en œuvre de la méthode

2.1 Plan du programme principal

La résolution va respecter les étapes vues précédemment.

Dans un premier temps on effectue une copie des matrices A et Y, de sorte à ce qu'elles ne soient pas modifiées après l'appel de la fonction principale **resolution** que l'on va coder. On rappelle que les tableaux présentent une particularité en Python à savoir que l'instruction A=B ne crée pas une copie du tableau B dans un nouveau tableau A mais donne un deuxième nom au tableau A ; de sorte à ce qu'une modification de A entraîne également une modification de B.

Ensuite l'objectif sera de rendre A triangulaire ; nous nous appuierons sur un certain nombre de fonctions, détaillées dans les paragraphes suivants.

Enfin, la phase de remontée sera réglée à l'aide d'une boucle, en utilisant l'algorithme vu au premier paragraphe. On prendra soin de retourner une matrice colonne ; que l'on pourra initialiser à l'aide de l'instruction suivante :

```
X=[0.]*n
```

Nota Bene : le programme complet "corrigé" sera distribué après les TPs

2.2 Fonctions auxiliaires

On va utiliser les fonctions suivantes :

- **ChercherPivot** : cette fonction prendra en entrée une matrice A , indice i . Elle retournera l'indice j tel $j \geqslant i$ et $|a_{j,i}|$ soit maximal.
- **EchangeLignes** : cette fonction permettra d'échanger deux lignes d'une matrice, que ce soit la matrice carrée A ou la matrice colonne Y. Elle prend en entrée une matrice A, les indices i et j des lignes à échanger ; et par souci de simplicité le nombre de colonnes nc de la matice A (même si il est possible de le recalculer à chaque fois). Cette fonction n'aura pas de retour, dans la mesure où la matrice A qui lui est passée sera directement modifiée.
- **Transvection** : cette fonction permettra de supprimer la i-ème variable à la ligne j>i. Elle prendra en entrée la matrice A, les indices i et j, le coefficient μ et nc le nombre de colonnes, toujours par souci de facilité.

2.3 Utilisation de numpy

On rappelle que numpy est une bibliothèque pour le calcul scientifique, optimisée pour les calculs sur les tableaux du type array.

Numpy propose une fonction de résolution d'équations linéaires avec la méthode de Gauss (à tester en TP) avec la syntaxe suivante :

```
numpy.linalg.solve(A,Y)
```

Attention Y est une matrice colonne ; par exemple [[1],[2],[3]].

3 Complexité

- La mise sous forme triangulaire :
  - Pour chaque valeur de $i \in [|0, n-2|]$ combien y-a-t-il de comparaisons pour la recherche du pivot ? D'affectations pour l'échange de deux lignes ?
  - Pour chaque valeur de $m \in [|i+1, n-1|]$ combien d'opérations coûte une transvection ?

  Pour chaque i recherche du pivot et échange de lignes ont complexité linéaire en n ainsi que chacune des transvections (il y en a au plus n) d'où complexité quadratique en n^2 Puisque, $i \in [|0, n-2|]$ la complexité est en $O(n^3)$
- Montrer que la complexité de la phase de remontée est en $O(n^2)$

Conclusion : L'algorithme du pivot de Gauss a une complexité en $O(n^3)$