Cours	Cours 11	TSI2
	Systèmes d'équations linéaires	Période 3
		1h

Principales capacités développées :
- réaliser un programme complet structuré allant de la prise en compte de données expérimentales à la mise en forme des résultats permettant de résoudre un problème scientifique donné,
- étudier l'effet d'une variation des paramètres sur le temps de calcul, sur la précision des résultats, sur la forme des solutions pour des programmes d'ingénierie numérique choisis, tout en contextualisant l'observation du temps de calcul par rapport à la complexité algorithmique de ces programmes,
- utiliser les bibliothèques standard pour afficher les résultats sous forme graphique,
- tenir compte des aspects pratiques comme l'impact des erreurs d'arrondi sur les résultats, le temps de calcul ou le stockage en mémoire.

La résolution de systèmes d'équations est indispensable aux problèmes spatiaux : fermeture géométrique, équilibre d'un solide soumis à des actions mécaniques…

On se propose d'étudier l'implémentation de la résolution d'un système d'équations par le pivot de Gauss pour un système de Cramer (autant d'inconnues que d'équations).

Exemple S : $\begin{cases} x + 2y = 1 \ \#(L1) \\ 3y + z = 4 \ \#(L2) \\ x + 5z = 6 \ \#(L3) \end{cases}$

On admet que ce système est de Cramer.

La bibliothèque utilisée pour ces calculs est **numpy** (permet un affichage lisible des tableaux de nombres et des fonctionnalités intéressantes pour ces opérations).

1 Implémentation du système d'équations

Pour numériser le système d'équations S, on écrit le système sous forme de tableaux.
M.X = B

Exemple $M = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 1 \\ 1 & 0 & 5 \end{bmatrix}$ $X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ $B = \begin{bmatrix} 1 \\ 4 \\ 6 \end{bmatrix}$

Pour la résolution du système d'équations, on passe ainsi par la matrice augmentée $Ma = [M \quad | \quad B]$.

Exemple $M_a = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 3 & 1 & 4 \\ 1 & 0 & 5 & 6 \end{bmatrix}$

Avec la bibliothèque **numpy**, on passe par un tableau à 2 dimensions contenant des flottants (les éléments sont typés dans **numpy** et les opérations sur les nombres pouvant conduire à des flottants il est préférable de créer un tableau de flottants).

```
Ma = np.array([[1,2,0,1],[0,3,1,4],[1,0,5,6]] , float)
```

2 Opérations sur les lignes

2.1 Extraction d'une ligne

Extraction de la i^e ligne du tableau **Ma** : **Ma [i]**

Exemple $M_a = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 3 & 1 & 4 \\ 1 & 0 & 5 & 6 \end{bmatrix}$

```
>>> Ma[1,:]
➔ array([ 0.,  3.,  1.,  4.])
```

2.2 Multiplication d'une ligne par un nombre

Multiplier la i^e ligne du tableau **Ma** par un nombre **a** (pivot à priori non nul) : **Ma[i] = a*Ma[i]**

2.3 Echanges entre 2 lignes

```
def echanger(M,p,q):
    if p!=q:
        Lq=M[q].copy()
        M[q]=M[p].copy()
        M[p]=Lq.copy()
    return M
```

fonctionne pour les listes et les **array** si **numpy** est chargé.
M : matrice, p et q sont le numéro des lignes à échanger
Extraction de la qiéme ligne de **M** stockée dans **Lq**
sans **copy()** alors **M[q]** est toujours un alias de **Lq** avec **numpy**

Exemple : échange entre les 2 premières lignes de $M_a = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 3 & 1 & 4 \\ 1 & 0 & 5 & 6 \end{bmatrix}$

```
>>> echanger (Ma,0 ,1)
>>> Ma
    ➔ [[0, 3, 1, 4],
       [1, 2, 0, 1],
       [1, 0, 5, 6]]
```

2.4 Combiner 2 lignes

Combiner 2 lignes p et q du tableau M en multipliant la ligne p par un nombre **a** non nul (Lq ←Lq+a*Lp)

M[q] = M[q] + a*M[p] # stocker sur la ligne q : (ligne q) + a * ligne p

Exemple : fin d'échelonnage de la colonne gauche de $M_{an} = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 3 & 1 & 4 \\ 1 & 0 & 5 & 6 \end{bmatrix}$

```
>>> Ma[2] = Ma[2] - Ma[0]    # L2 ←L2+(-1)*L0
>>> Ma
array ([[1, 2, 0, 1],
        [0, 3, 1, 4],
        [0, -2, 5, 5]])
```

$$\begin{cases} x + 2y = 1 \ \#(L0) \\ 3y + z = 4 \ \#(L1) \\ -2y + 5z = 5 \ \#(L2) \leftarrow (L2 - L0) \end{cases}$$

2.5 Extrait une colonne

Extraire la i^e colonne d'un tableau M : **M[:,i]**
Extraire la dernière colonne d'un tableau M : **M[:,-1]**

3 Résolution du système d'équations

Pour résoudre numériquement un système d'équations, il faut obtenir une matrice échelonnée réduite :

3.1 Echelonner

```
[[ 1.        2.        0.          1.        ]
 [ 0.        3.        1.          4.        ]
 [ 0.        0.        5.66666667  7.66666667]]]
```

$$\begin{cases} x+2y=1 & \#(L1) \\ 3y+z=4 & \#(L2) \\ 5.67z=7.67 & \#(L3) \leftarrow (2.L2/3+L3) \end{cases}$$

3.2 Réduire

```
[[ 1.        2.        0.          1.        ]
 [ 0.        1.        0.33333333  1.33333333]
 [ 0.        0.        1.          1.35294118]]
```

$$\begin{cases} x+2y=1 & \#(L1) \\ y+0.33z=1.33 & \#(L2) \\ z=1.35 & \#(L3) \leftarrow (L3/5.66) \end{cases}$$

3.3 Remonter

```
[[ 1.        0.        0.        -0.76470588]
 [ 0.        1.        0.         0.88235294]
 [ 0.        0.        1.         1.35294118]]
```

$$\begin{cases} x=-0.761 & \#(L1) \\ y=0.88 & \#(L2) \\ z=1.35 & \#(L3) \end{cases}$$

3.4 Récupérer la solution du problème = dernière colonne de Ma

[-0.76470588, 0.88235294, 1.35294118]

4 Erreurs d'arrondis et pivot partiel

Dans le cas, où le pivot est petit des erreurs d'arrondis peuvent conduire à des solutions erronées.

Exemple : S2 : $\begin{cases} 10^{-4}x+y=1 \\ x+y=2 \end{cases}$ → solution exacte : x = 1.0001 et y=0.9999

Résolution en virgule flottante à 3 chiffres significatifs :

- pivot petit $M[0,0]$ → erreur d'arrondi possible
  $L_2 \leftarrow L_2 - 10^4 L_1$ → $-9999y=-9998$ soit $9.99*10^3 y = 9.99*10^3$ avec 3 chiffres d'où $y=1$.
  en reportant dans L_1 : $x=0$

- pivot grand $M[1,0]$ → limite les erreurs d'arrondi
  $L_1 \leftarrow L_1 - 10^{-4} L_2$ → $-0{,}9999y=-0{,}9998$ avec 3 chiffres significatifs → $y=1$.
  en reportant dans L_2 : $x=1$

Cet exemple illustre qu'il est préférable d'utiliser le pivot le plus grand possible afin de limiter l'impact des erreurs d'arrondis.
La méthode du **pivot partiel** consiste à choisir pour une colonne i donnée le pivot le plus grand possible en valeur absolue soit le coefficient $|M[i,j]|$ **maximum** avec $i \le j \le n$.
Une fois le pivot partiel trouvé, il convient d'échanger les lignes **i** et **j** pour retrouver un pivot **M[i,i]**.

Remarque : il existe un pivot total maximum sur tout le tableau mais cela implique de changer la position des variables avec toutes les complications induites.

ANNEXE Tableaux avec la bibliothèque **numpy**

Import de la bibliothèque numpy par :
import numpy as np

Fonctions nécessitant le préfixe **np** :

Créer un vecteur (tableau à 1 dimension)	v=np.array([1, 2, 3])
Créer un tableau colonne (2 dimensions : 1 colonne et 3 lignes)	v=np.array([[1],[2],[3]])
Tableau à 2 dimensions	M=np.array([[1,2,3],[4,5,6]])
Séquence équirépartie quelconque	np.arange(0,10.1,0.1)
Tableau de zéros (avec des flottants)	np.zeros((2,3),float)
Tableau de uns (avec des flottants)	np.ones((2,3),float)
Produit matriciel	np.dot(v,w)
Résolution de système matriciel M.X=Y	np.linalg.solve(M,Y)

Instructions sans préfixe **np** car s'appliquant aux tableaux (**array**)

Accéder à un élément	v[0], M[0,1]
Accéder au dernier élément, et l'avant dernier	v[-1], v[-2]
Dimensions d'un tableau	M.shape
Extraire la 2ème ligne	M[1,:] ou M[1]
Extraire la 2ème colonne	M[:,1]
Extraire une portion de tableau (2 premières colonnes)	M[:,0:2]
Extraire des éléments d'un tableau par leurs indices	M[0,(2,1)]
Copier un tableau dans une autre variable	w=v.copy()
Maximum et minimum d'un tableau	v.max(0), v.min(0)
Indice i du maximum	v.argmax(0)
Transposée	M.transpose()

Fonction **python**

Séquence équirépartie d'entiers	range(1,11)