Variations autour du pivot de Gauss

D'après un TP de Stéphane Gonnord

Buts du TP

- Mettre en place l'algorithme complet de résolution d'un système linéaire par pivot de Gauss.
- Observer les limitations de cet algorithme dans le monde numérique.
- Mettre en place des algorithmes utilisant le même principe de pivot.

L'objectif principal est que tout le monde ait codé l'algorithme du pivot. La deuxième partie est ici à titre de compléments, pour ceux qui veulent approfondir. Dans le cadre de ce TP on se placera toujours dans le cas de résolution d'un système de Cramer : système linéaire de n équations à n inconnues qui comporte une unique solution (la matrice carré du système est inversible).

EXERCICE 1 *Copier le répertoire* `TP11-Gauss` *contenu dans le répertoire* `Donnees` *de la classe et le coller dans son répertoire personnel. Le fichier* `cadeau.py` *contient une fonction de copie de matrice, une fonction pour générer une matrice carrée identité et quelques compléments sur les expressions booléennes et leur interêt pour construire des matrices comme la matrice identité ou la matrice de Virginie (voir exo 14).*

Lancer `Pyzo` *ou* `Spyder`*, créer puis sauvegarder immédiatement au bon endroit le fichier* `TP11.py`.

EXERCICE 2 *Résoudre à la main les systèmes suivants :*

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

1 Résolution d'un système

Pour résoudre le système $Ax = y$, on va effectuer des opérations sur les lignes de la matrice A étendue avec comme colonne supplémentaire la matrice colonne y.

On commence par mettre le système sous forme triangulaire :

pour i *de* 0 *à* $n-2$ **faire**
  Trouver j entre i et $n-1$ tel que $|a_{j,i}|$ soit maximale. # «pivot partiel»
  # pour l'algorithme de base, on se contente de trouver un terme non nul.
  Échanger L_i et L_j (coefficients de la matrice **et** membres de droite).
  pour k *de* $i+1$ *à* $n-1$ **faire**
    $L_k \leftarrow L_k - \dfrac{a_{k,i}}{a_{i,i}} L_i$ # mettre à zéro les coefficients en position (k,i)

Arrivé ici, le système est sous forme triangulaire et il n'y a plus qu'à « remonter », via des substitutions. Le résultat est mis dans une liste/tableau x, et il s'agit donc de calculer :

$$x_i = \frac{1}{a_{i,i}} \left(y_i - \sum_{k=i+1}^{n-1} a_{i,k} x_k \right).$$

EXERCICE 3 *Si vous n'aviez pas fait comme ça, reprenez les exemples de l'exercice 2 en utilisant le pivot de Gauss !* Inutile de prendre un pivot de valeur absolue maximale : prenez-le simplement non nul !

EXERCICE 4 *Sans aller voir son cours, écrire une fonction d'échange de ligne dans une matrice, et une fonction de transvection.* Attention, ces fonctions ne renvoient rien : elles modifient en place les matrices données en argument.

```
def echange_ligne(A, i, j):
    ...
def transvection(A, i, j, mu): # Li <- Li + mu.Lj
    ...
>>> A = [[4, 5, 6], [1, 2, 3]]
>>> echange_ligne(A, 0, 1)
>>> A
[[1, 2, 3], [4, 5, 6]]
>>> transvection(A, 1, 0, -4)
>>> A
[[1, 2, 3], [0, -3, -6]]
```

EXERCICE 5 *Écrire une fonction chargée de la recherche du pivot de module maximal dans une matrice sur une colonne (disons j_0) à partir de la ligne j_0 : il s'agit de renvoyer le (un) $i \geqslant j_0$ tel que $|A_{i,j_0}|$ est maximal.*

```
def pivot_partiel(A, j0):
   ...
   return i
>>> A = [[1, 2, 3, 4], [0, 1, 3, 5], [0, -4, 1, 0], [0, 3, 0, 0]]
>>> pivot_partiel(A, 1)
2
```

EXERCICE 6 *Toujours sans regarder son cours, écrire une fonction réalisant la résolution d'un système linéaire par pivot de Gauss (en supposant ledit système de Cramer ; si cette hypothèse n'est pas vérifiée, aucun comportement n'est spécifié). On travaillera sur des copies de la matrice* `A` *et des ordonnées* `y0`*. une fonction de copie de matrice se trouve dans le fichier* `cadeau.py`.

```
def resolution_systeme(A0, y0):
    A = copie_matrice(A0) # pour ne pas modifier la matrice donnée en paramètre.
    y = copie_matrice(y0)
    ...
    return x
```

EXERCICE 7 *Tester le programme sur les exemples de l'exercice 2*

```
>>> resolution_systeme([[2, 3],[5,-2]],[[5], [-16]])
[[-2.0], [3.0]]
```

EXERCICE 8 *Comparer les résultats avec ceux obtenus à l'aide de la fonction* `solve` *de la librairie* `numpy.linalg`

EXERCICE 9 *Résoudre les systèmes suivants de trois façons (avec votre fonction de résolution, à la main, et avec* `numpy.linalg.solve`*), puis commenter :*

$$\begin{cases} x+2y &= 1 \\ 2x+4y &= 1 \end{cases} \quad et \quad \begin{cases} x+2y &= 1 \\ 2x+4y &= 2 \end{cases}$$

EXERCICE 10 *Résoudre (de trois façons...) les systèmes suivants, puis commenter :*

$$\begin{cases} x + 1/4y + z = 0 \\ x + 1/3y + 2z = 0 \\ y + 12z = 1 \end{cases} \quad et \quad \begin{cases} x + 10^{15}y + z = 1 \\ x + 10^{-2}y + 2z = 0 \\ 10^{15}y - z = 0 \end{cases}$$

2 Inversion d'une matrice

L'algorithme du pivot nous a amené à faire des opérations sur les lignes d'une matrice, ce qui revient à la multiplier à gauche par des matrices élémentaires, disons $L_1, ..., L_N$ (après N opérations).

Le système $AX = Y$ est alors équivalent à $L_1AX = L_1Y$ puis $L_2L_1AX = L_2L_1Y$, puis $L_N...L_1AX = L_N...L_1Y$, et si on a correctement appliqué l'algorithme du pivot [1], le membre de gauche vaut exactement X. Ainsi : $A^{-1} = L_N...L_1$. Cette matrice est en fait exactement celle obtenue en appliquant à la matrice identité (et dans le même ordre) les opérations réalisées sur A.

EXERCICE 11 *Mettre en œuvre cette idée pour calculer l'inverse d'une matrice : expliciter l'algorithme suggéré dans les lignes précédentes. Écrire le programme Python correspondant à l'algorithme.*

```
def inversion(A0):
    A = copie_matrice(A0) # pour ne pas modifier la matrice donnée en paramètre.
    ...
    return A
```

Vérifier :

$$\begin{pmatrix}1&2\\0&3\end{pmatrix}^{-1} = \begin{pmatrix}1&-\frac{2}{3}\\0&\frac{1}{3}\end{pmatrix} \quad \begin{pmatrix}1&2\\3&5\end{pmatrix}^{-1} = \begin{pmatrix}-5&2\\3&-1\end{pmatrix} \quad \begin{pmatrix}1&2\\3&4\end{pmatrix}^{-1} = \begin{pmatrix}-2&1\\\frac{3}{2}&-\frac{1}{2}\end{pmatrix}$$

$$\begin{pmatrix}2&2&-3\\0&1&-6\\0&0&1\end{pmatrix}^{-1} = \begin{pmatrix}\frac{1}{2}&-1&-\frac{9}{2}\\0&1&6\\0&0&1\end{pmatrix} \quad \begin{pmatrix}2&2&-3\\-2&-1&-3\\6&4&4\end{pmatrix}^{-1} = \begin{pmatrix}4&-10&-4,5\\-5&13&6\\-1&2&1\end{pmatrix}$$

EXERCICE 12 *Déterminer l'inverse des matrices suivantes :*

$$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix} \quad \begin{pmatrix}1&1/4&1\\1&1/3&2\\0&1&12\end{pmatrix} \quad \begin{pmatrix}1&1+10^5&1\\1&1+10^{-5}&2\\0&10^5&-1\end{pmatrix} \quad \begin{pmatrix}1&10^{15}&1\\1&10^{-2}&2\\0&10^{15}&-1\end{pmatrix}$$

EXERCICE 13 *Sur les exemples précédents, comparer le résultat obtenu avec celui fourni par la fonction* `inv` *de la bibliothèque* `numpy.linalg`.

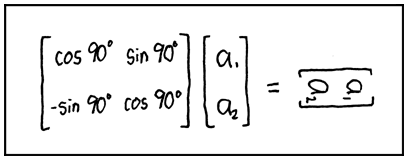
EXERCICE 14 *La matrice de Virginie d'ordre n est la matrice $V \in \mathcal{M}_n(\mathbb{R})$ suivante :*

$$V = \begin{pmatrix}2&-1&&&(0)\\-1&2&-1&&\\&\ddots&\ddots&\ddots&\\&&-1&2&-1\\(0)&&&-1&2\end{pmatrix}$$

1. *Inverser V_2 et V_3.*
2. *Évaluer le temps de calcul nécessaire pour inverser V lorsque $n \in \{50, 100, 200, 400\}$. Comparer avec les temps équivalents nécessaires à* `numpy.linalg.inv`.

```
import time

t0 = time.time()
... [mon calcul] ...
t1 = time.time()  # t1-t0 est le temps (en secondes) du calcul
```



1. ce qui inclut la remontée : les substitutions correspondent à des transvections $L_i \leftarrow L_i - \alpha L_k$, et on termine avec des dilatations $L_i \leftarrow \frac{1}{a_{i,i}} L_i$