

FEUILLE DE TRAVAUX DIRIGÉS 3**1. INVERSION D'UNE MATRICE**

Notons $(e_1, \dots, e_n)$ la base canonique de $\mathbb{R}^n$ pour un entier $n \geq 1$. On notera en colonne les vecteurs de $\mathbb{R}^n$.

Considérons une matrice $A \in GL_n(\mathbb{R})$. Pour $1 \leq i \leq n$, la i -ème colonne de A^{-1} est égale au vecteur $u_i = A^{-1}e_i$. On a ainsi le système suivant :

$$A \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = e_i \quad \text{où} \quad u_i = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad \text{et} \quad e_i = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix},$$

le 1 se trouvant sur la i -ème ligne.

Il suffit donc de résoudre n systèmes linéaires pour calculer l'inverse de A .

Dans la pratique, on résout simultanément ces n systèmes linéaires. Pour cela, on applique la méthode du pivot de Gauss en travaillant en même temps sur l'ensemble des seconds membres.

1.1. Étapes de l'algorithme.

- (1) Afin de représenter simultanément les n systèmes, on adjoint tout d'abord à A les n colonnes correspondant aux seconds membres (c'est-à-dire la matrice identité).

Par exemple si $n = 3$, on considère :

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & 0 & 0 & 1 \end{pmatrix}$$

- (2) En utilisant des pivots de Gauss récursivement sur les n colonnes de A , on triangularise les n systèmes.

Pour $n = 3$, on obtient n systèmes représentés par :

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & b_{11} & b_{12} & b_{13} \\ 0 & a_{22} & a_{23} & b_{21} & b_{22} & b_{23} \\ 0 & 0 & a_{33} & b_{31} & b_{32} & b_{33} \end{pmatrix}$$

- (3) Pour $1 \leq i \leq n$, on divise chaque ligne i par a_{ii} de façon à obtenir des 1 sur la diagonale. (Les a_{ii} sont tous non nuls, sinon la matrice de départ ne serait pas diagonalisable.)

Dans le cas $n = 3$, on obtient :

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & b'_{11} & b'_{12} & b'_{13} \\ 0 & 1 & a'_{23} & b'_{21} & b'_{22} & b'_{23} \\ 0 & 0 & 1 & b'_{31} & b'_{32} & b'_{33} \end{pmatrix}$$

- (4) Pour terminer, on transforme les lignes afin d'amener des 0 au-dessus de la diagonale. Pour cela, on retranche le multiple par a'_{ij} de la ligne j à la ligne i , puis un multiple adéquat de la ligne k à la ligne i , et ainsi de suite (en pratique, afin de limiter le nombre de calculs, on commence par la dernière colonne).

Pour $n = 3$, on a alors :

$$\begin{pmatrix} 1 & 0 & 0 & b''_{11} & b''_{12} & b''_{13} \\ 0 & 1 & 0 & b''_{21} & b''_{22} & b''_{23} \\ 0 & 0 & 1 & b''_{31} & b''_{32} & b''_{33} \end{pmatrix}$$

2. EN TD

- (1) Dérouler l'algorithme d'inversion d'une matrice décrit dans la première partie sur l'exemple suivant :

$$A = \begin{pmatrix} 0 & 1 & 3 \\ 1 & 0 & -2 \\ 2 & 1 & 1 \end{pmatrix}$$

- (2) Quelle est la complexité de cet algorithme ?
 (3) En utilisant une partie de l'algorithme d'inversion, donner un algorithme permettant de calculer le déterminant d'une matrice.
 (4) Préparer le TP décrit dans la partie suivante.
 (5) Question facultative : si l'on considère uniquement des matrices à coefficients rationnels, comment peut-on implémenter cet algorithme avec des calculs exacts (et non des valeurs approchées) ?

3. EN TP

Le but de ce TP est d'implémenter une classe permettant d'inverser une matrice à l'aide du pivot de Gauss (algorithme décrit dans la première partie). Les matrices seront représentées par des tableaux à deux dimensions.

- (1) On commence par une fonction prenant en entrée la matrice à inverser et initialisant un tableau représentant les n systèmes linéaires à résoudre (cf. I.1).
 (2) Implémenter au fur et à mesure les étapes de l'algorithme décrit dans la première partie. Afin de tenir compte des erreurs de calculs, on fixera une constante valant 10^{-8} au-dessous de laquelle les valeurs seront considérées comme nulles. Vous êtes encouragés à écrire plusieurs fonctions.
 (3) Créer une fonction **inverse** qui prend en entrée une matrice A et retourne son inverse. Elle combinera les fonctions de la question précédente.
 (4) Ajouter une fonction retournant le déterminant d'une matrice passée en argument.
 (5) Créer une fonction inversant une matrice de rationnels en utilisant le package **fractions**.
 (6) Bonus : Lever des exceptions (syntaxe `try except`) lorsque l'inversion n'est pas possible (matrice non carrée par exemple).