Pivot de Gauss

L'objectif de ce TP est de rédiger un programme réalisant un pivot de Gauss. La première partie vous permettra d'appliquer les notions du chapitre concernant l'allocation dynamique de matrices. La seconde partie est plus algorithmique et consiste à implémenter l'algorithme vu en cours d'algèbre. Votre code final sera alors composé de deux bibliothèques :

- une librairie manipulant les matrices (lecture, affichage, écriture) ;
- une librairie réalisant un pivot de Gauss.

Todo. Créez les fichiers vides *matrice.c/.h* et *gauss.c/.h* ainsi qu'un fichier *main.c*.

1 Bibliothèque de manipulation de matrices

Cette première partie consiste à compléter la librairie *matrice.c/.h*. L'objectif est d'éviter à l'utilisateur la saisie fastidieuse de tous les coefficients de la matrice dans le programme. Il est plus simple de rédiger au préalable un fichier texte contenant ces coefficients. Votre programme devra donc être capable le lire un fichier contenant une matrice, afficher ses coefficients et écrire la matrice échleonnée réduite dans un nouveau fichier.

La matrice doit être stockée par l'utilisateur dans un fichier texte de la façon suivante :

- la première ligne contient deux valeurs entières séparées par une tabulation : le nombre de lignes m suivi du nombre de colonnes n de la matrice ;
- sur m lignes : n valeurs séparées par des tabulations.

Exemple 1. Le code suivant permet de décrire une matrice 2×4 d'entiers.

```
2   4
11  12  13  14
21  22  23  24
```

Votre programme doit donc dans un premier temps lire ce fichier pour récupérer les valeurs de la matrice. La lecture utilise les fonctions vues dans le cours sur les fichiers : `fopen` et `fclose`. La fonction de lecture est alors `fscanf` car les éléments à lire sont des nombres. L'allocation de la matrice doit bien évidemment être réalisée de façon dynamique en utilisant la fonction `calloc`.

Todo. Créez dans un premier temps la fonction `allocMatrice` permettant d'allouer une matrice aux dimensions spécifiées en paramètres. Par la suite, complétez la fonction `lireMatrice` afin de réaliser les actions suivantes. Une fois ouvert, lisez les deux premières valeurs du fichier passé en paramètres afin de mettre à jour les dimensions de la matrices passées par adresse. Allouez correctement la matrice puis poursuivez la lecture du fichier pour remplir la matrice. Votre code doit retourner un pointeur sur la matrice ainsi allouée et remplie. Rédigez enfin une fonction `afficherMatrice` permettant d'afficher une matrice.

2 Méthode de Gauss pas à pas

Dans cette seconde partie, votre objectif est de réaliser un programme d'aide à la réalisation d'un pivot de Gauss. Pour cela, vous devrez coder les fonctions `transposition()`, `dilatation()` et `transvection()` (le détail de chacune de ces fonctions vous est donné plus bas). L'action utilisateur attendue pour appliquer les opérations sera la saisie d'une lettre référençant l'opération : `'E'` pour transposition (*échange* de deux lignes), `'D'` pour dilatation, `'T'` pour transvection. Il faut donc tout d'abord réaliser un menu pour afficher les possibilités à l'utilisateur, et récupérer son choix.

2.1 Interface utilisateur

Afin de faciliter la tâche à l'utilisateur, votre programme doit dans un premier temps afficher la matrice sur le terminal. Vous devez ensuite demander à l'utilisateur quelle action il souhaite réaliser (`EDT`). Suivant sa demande, vous récupérez alors deux ou trois valeurs qui vous serviront à effectuer l'action demandée.

ATTENTION : la première valeur d'une matrice en maths est en position $(1, 1)$, en C, cette même position est notée $(0, 0)$! L'utilisateur saisit bien évidemment les indices en langage mathématique !

Todo. `int gaussManuel (double **M, int m, int n);`

Rédigez la partie interface dans cette nouvelle fonction `pivot`. Concentrez-vous pour le moment sur la partie interface utilisateur, c'est à dire afficher la matrice et récupérer l'action utilisateur, avec les indices de lignes voulus par ce dernier.

2.2 Transposition

La *transposition* consiste à échanger deux lignes de la matrice. Pour simplifier l'appel à cette fonction, les arguments seront au nombre de trois. Vous utiliserez donc deux pointeur sur les lignes à échanger ainsi que la taille d'une ligne.

Todo. `int transposition (double *l1, double *l2, int n);`

Rédigez la fonction ci-dessus échangeant les `n` valeurs présentes dans les lignes `l1` et `l2`. Prenez soin de tester au préalable la validité de ces deux pointeurs.

2.3 Dilatation

Cette fonction consiste à multiplier chaque valeur d'une ligne par un scalaire donné. La fonction doit donc prendre un pointeur sur la ligne à modifier ainsi que la valeur du scalaire. Encore une fois, pensez à vérifier la validité de l'ensemble des arguments !

Todo. `int dilatation (double *l, int n, doube lambda);`

Rédigez la fonction `Dilatation` permettant de multiplier chaque valeur de la ligne `l` par le scalaire `lambda`.

2.4 Transvection

L'opération de *transvection* consiste à ajouter à une ligne une autre ligne multipliée par un certain coefficient. Pour effectuer l'opération $L_i \leftarrow L_i + \lambda L_j$, votre fonction a donc

besoin de trois paramètres : un pointeur sur chacune des lignes ainsi que le scalaire λ.

Todo. `int transvection (double *li, double *lj, int n, double lambda);`

Cette fonction réalise l'opération de transvection présentée précédemment. N'oubliez pas de tester les pointeurs !

2.5 Réalisation du pivot de Gauss en manuel

Il est enfin temps d'utiliser les fonctions de manipulation de la matrice pour la réalisation du pivot de Gauss. Vous devez donc compléter la fonction `gaussManuel` qui réalise les actions suivantes tant que l'utilisateur le désire :

1. Nettoyage de l'écran (`system("clear");`).
2. Affichage de la matrice et du menu
3. Récupération de l'action utilisateur
4. Réalisation de l'action demandée sur la matrice

Todo. À vous de trouver le moyen de continuer cette boucle tant que l'utilisateur le désire. Vous pourrez par la suite ajouter quelques fonctionnalités supplémentaires telles que l'annulation de la dernière action ou la ré-initialisation de la matrice.

3 Pour aller plus loin : méthode de Gauss automatisée

Cette dernière partie vous permettra de réaliser la méthode du pivot de Gauss puis la réduction de la matrice.

Algorithme 1 – Pivot de Gauss
Entrée. Matrice de réels M[n][p]
Sortie. Matrice de réels M échelonnée

```
Entier Li ← 1
Pour j allant de 1 à p faire
    Recherche de e: premier élément non nul de la colonne j à partir de
      Li
    Entier pivotVal ← e
    Entier pivotInd ← ligne(e)
    Si un pivot a été trouvé alors
        Échanger la ligne pivotInd avec la ligne Li
        Pour i allant de Li+1 à n faire
            Transvection(M, i, Li, -M[i][j]/M[Li][j])
        Fin Pour
        Li ← Li + 1
    Fin Si
Fin Pour
```

3.1 Automatisation

L'algorithme présenté ci-dessus réalise l'automatisation du pivot de Gauss. Pour le moment, on ignore l'étape de réduction permettant de mettre à 0 les valeurs au-dessus des pivots.

Todo. `int gaussAuto (double **M, int m, int n);`

À vous de coder cet algorithme dans la fonction `gaussAuto`.

3.2 Réduction

La fonction `reductionGauss` doit vous permettre de réaliser une réduction de la matrice une fois la matrice échelonnée obtenue. C'est à vous de trouver l'algorithme de cette dernière fonction. N'oubliez pas de rechercher à nouveau les pivots sur chaque ligne à partir de la dernière.

Todo. Rédigez l'algorithme dans un premier temps sur papier puis implémentez-le. N'hésitez pas à relire le cours d'algèbre !

4 Exemples à tester

Vous trouverez dans cette partie quelques exemples afin de tester le bon fonctionnement de vos algorithmes. N'hésitez pas à effectuer d'autres tests ! Considérons les matrices M_1, M_2 et M_3 suivantes.

$$M_1 = \begin{pmatrix} 3 & 2 & -1 & 12 \\ 2 & -4 & 1 & -1 \\ -4 & 1 & 2 & -8 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 4 & 5 & 6 & 7 \\ 0 & 6 & 7 & 8 & 9 \end{pmatrix}, \quad M_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 0 & 0 & 1 & -1 \\ -2 & -2 & 0 & 0 & 3 & 1 \\ 0 & 0 & 1 & 1 & 3 & -1 \\ 1 & 1 & 2 & 2 & 4 & 1 \end{pmatrix}$$

Leurs matrices échelonnées réduites respectives sont les suivantes.

$$M_1' = \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \quad M_2' = \begin{pmatrix} 0 & 1 & 0 & -1 & -2 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ et } M_3' = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$