Université de Nice-Sophia Antipolis — Avril 2023
ELEC3 — Durée : 2 mois

Projet C : Résolution de système d'équations linéaires

Avant de commencer : la qualité des commentaires, avec notamment la présence des antécédents, des conséquents, des invariants de boucle, les rôles de chacune des fonctions, ainsi que les noms donnés aux variables, l'emploi à bon escient des majuscules et la bonne indentation rentreront pour une part importante dans l'appréciation du travail. Ce projet doit permettre de montrer votre autonomie et votre compréhension tant dans la conception du programme que dans sa réalisation. Enfin, si les codes de plusieurs projets se trouvent être identiques, ou être copiés depuis le web, tous les projets concernés seront immédiatement sanctionnés par un zéro.

Objectif : mettre en œuvre la méthode du pivot de Gauss [1], pour résoudre un système linéaire de N équations à N inconnues, en utilisant deux méthodes de représentation des matrices : les matrices *simples* et les matrices *creuses*. Le système sera visualisé à l'aide d'une interface graphique.

1 Résolution d'un système linéaire à N inconnues

La résolution d'un système linéaire par la méthode du pivot de Gauss se fait en deux étapes. La première transforme la matrice du système en une matrice triangulaire avec des 1 sur sa diagonale. La seconde calcule les solutions du système, à partir de la matrice triangularisée, en partant de la dernière équation jusqu'à la première.

Ainsi le système linéaire suivant :

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}$$

est transformé, par la méthode du pivot de Gauss, en le système équivalent :

$$\begin{pmatrix} 1 & a'_{12} & \dots & a'_{1N} \\ 0 & 1 & \dots & a'_{2N} \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_N \end{pmatrix}$$

Pour triangulariser une matrice A, on procède de façon itérative du rang 1 au rang N. À la k^e étape, la sous-matrice $(k-1, k-1)$ est triangularisée et les opérations à effectuer pour poursuivre la triangularisation sont les suivantes :

- choisir le pivot ;
- normaliser la ligne, c'est-à-dire diviser la ligne k du système par le pivot (*i.e.* les a_{kj} et b_k) ;
- pour toute ligne i du système allant de $k+1$ à N, lui soustraire la ligne de rang k multipliée par a_{ik}.

Pour diminuer les risques d'erreurs dans les calculs numériques, on choisit le pivot a_{jk} le plus grand en valeur absolue entre les lignes k et N. Notez qu'il faut alors permuter les lignes j et k si $j \neq k$. Remarquez que si on ne peut trouver de pivot différent de zéro, le système est *lié* et n'admet pas de solution unique.

1. C. F. Gauss, astronome, mathématicien et physicien allemand (1777-1855).

Notez aussi que ces opérations ne modifient évidemment pas la solution du système d'équations. Celle-ci se calcule par substitution « *en remontant* » à partir de la dernière équation : $x_N = b'_N$, $x_{N-1} = b'_{N-1} - a'_{N-1,N} x_N$, *etc.*. L'algorithme de Gauss est donné ci-dessous :

```
{Antécédent : A, B données du système linéaire}
{Conséquent : X solution du système AX=B}
GaussRSL(données  A : matrice, B : vecteur
         résultat X : vecteur
début
    {triangularisation de la matrice A}
    pourtout k de 1 à N faire
       pivot ← { prendre le pivot le + grand en valeur absolue }
       {normaliser la ligne k de la matrice A et}
       {du vecteur B par le pivot tel que A_kk = 1}
       ...
       pourtout i ← k+1 à N faire
           {soustraire à la i^e ligne de la matrice A et}
           {du vecteur B la ligne k multipliée par A_ik}
           ...
       finpour
    finpour
    {calcul de la solution X}
    pourtout k de N à 1 faire
       sol ← B_k
       {calcul de la solution sol par substitution en remontant}
       {jusqu'à la k+1^e ligne}
       ...
       X_k ← sol
    finpour
fin
```

2 Première implémentation : les matrices simples

La résolution du système nécessite la définition d'un type *matrice* et d'un type *vecteur*.

Dans cette première version, les matrices sont *simples*, c'est-à-dire qu'elles contiennent toutes les valeurs. Une matrice sera une structure avec trois champs définie comme suit :

```
typedef double T;

typedef struct {
  int nbl; // le nombre de lignes
  int nbc; // le nombre de colonnes
  T *mat;  // les éléments de la matrice
} matrice;
```

Les éléments de la matrice sont placés dans un tableau alloué *dynamiquement*.

En respectant la déclaration du type `matrice` précédent, vous écrirez les routines de manipulation de matrice suivantes :

```
// rôle : renvoie une matrice m × n initialisée à 0
extern matrice matriceNulle(const int m, const int n);
// rôle : renvoie l'élément m(i,j)
extern T getM(const matrice m, const int i, const int j);
// rôle : m(i,j) = x
extern void setM(matrice *m, const int i, const int j, const T x);
// rôle : permute les lignes i et j de la matrice m
```

```
extern void echangerLignesM(matrice *m, const int i, const int j);
// rôle : écrit la matrice m sur la sortie standard
extern void ecrireMatrice(const matrice m);
// rôle : initialise la matrice m à partir du fichier de nom f
extern void lireMatrice(matrice *m, const char *f);
// rôle : renvoie le nombre de lignes de la matrice m
extern int getNbLignes(const matrice m);
// rôle : renvoie le nombre de colonnes de la matrice m
extern int getNbCols(const matrice m);
```

En vous basant sur le type `matrice` précédent, définissez le type `vecteur` et ses routines de manipulation.

La définition des *matrices* et des *vecteurs* doit être faite dans les fichiers `matrice.c`, `matrice.h`, `vecteur.c` et `vecteur.h`.

---

1) Programmez la résolution du système d'équations linéaires selon la méthode de Gauss donnée plus haut. Dans un fichier `main.c`, vous écrirez la fonction `GaussRSL` dont l'en-tête, à respecter, est le suivant :

```
void GaussRSL(matrice A, vecteur B, vecteur *X)
```

Dans le fichier `main.c`, vous écrirez la fonction `main` pour testez votre procédure `gaussRSL`. Le vecteur solution sera écrit sur la sortie standard.

Testez votre procédures avec plusieurs matrices et vecteurs différents.

Une matrice et un vecteur seront initialisés à partir de données lues dans un fichier de texte. Un fichier pour une matrice contiendra en 1ère ligne les dimensions de la matrice, suivie des valeurs de la matrice. *Idem* pour le vecteur. Ci-dessous, deux fichiers qui contiennent, respectivement, une matrice 3×4 et un vecteur de dimension 5 :

un fichier pour une matrice 3×4

```
3 4
1.0 2.9 3.7 4.0
9.1 8.0 7.8 6.87
4.2 3.0 2.8 1.99
```

un fichier pour un vecteur de dimension 5

```
5
18.9 1.0 9.1 8.0 7.8
```

3 Seconde implémentation : les matrices creuses

Dans cette méthode, on désire définir une représentation des matrices qui économise de la place mémoire. On trouve un peu dommage de garder des éléments de valeur nulle, surtout s'il y en a beaucoup (c'est le cas de la matrice triangularisée). Les matrices que nous allons définir, sont appelées *matrices creuses*, car elles ne contiennent que des valeurs différentes de 0.

Une matrice creuse sera une structure composée de trois champs :

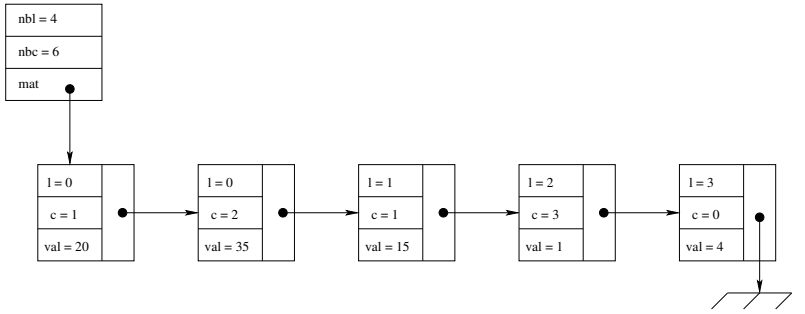
- `nbl`, le nombre de lignes de la matrice ;
- `nbc`, le nombre de colonnes de la matrice ;
- `mat`, les éléments de la matrice placés dans une « *suite* » de triplets. Chaque triplet contient l'indice i de la ligne, l'indice j de la colonne et l'élément (non nul) de la case (i, j) de la matrice.

La suite de triplets sera représentée par une liste linéaire telle que vue au TD 12. Vous ne devrez pas changer la déclaration du type `liste`[2].

Exemple :

$$\begin{pmatrix} 0 & 20 & 35 & 0 & 0 & 0 \\ 0 & 15 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

sa représentation en C sous forme d'une matrice creuse :



2) Programmez la mise en œuvre des matrices creuses dans les fichiers `matriceC.c` et `matriceC.h`.

3) Testez votre programme de résolution du système linéaire avec une matrice creuse.

TRÈS IMPORTANT : le fichier `main.c` **NE** doit **PAS** être modifié (à l'exception du `include`, puisque le nom du fichier à inclure change), ce qui veut aussi dire que les en-têtes des routines définies dans `matriceC.c`, `matriceC.h`, `vecteur.c` et `vecteur.h`, ne doivent pas l'être non plus !

4 Interface graphique

On veut maintenant visualiser graphiquement le système d'équations linéaires avec sa solution. Pour cela vous utiliserez la bibliothèque graphique *libsx*.

Vous pourrez faire deux versions de difficulté croissante :

1. L'interface permet de charger la la matrice et le vecteur à partir de fichiers à l'aide d'un menu, et visualise le système d'équation dans un fenêtre. Un bouton lance la résolution du système et visualise, toujours graphiquement, la solution.
2. *Idem* que précédemment, mais le calcul de la solution est visualisé graphiquement, *étape par étape*, pour montrer la progression de la résolution du système d'équations linéaires.

5 Remise du projet

Votre projet est à faire en binôme (voir liste des binômes) et vous devrez le rendre au plus tard :

2. Soit vous utilisez l'implémentation du type `liste` donnée en correction sur mon site web, soit vous utilisez celle qui conserve la longueur de la liste et gère un élément bidon (comme discuté en TD). Quoi qu'il en soit, les routines de manipulation de la liste doivent conserver les mêmes en-têtes.

le vendredi 9 juin 2023, 23h59 – aucun délai ne sera accordé –

sous forme d'une archive `gaussRSL.tgz` que vous déposerez sur le site de dépôt **Moodle/LMS UCA 22 : EIEL622 - ECUE Langage C-2**. Ne déposer qu'un seul projet par binôme.

Attention : ce site gère la date limite de rendu. Après ce sera trop tard. N'attendez pas la dernière minute, vous pouvez déposer des versions intermédiaires de votre projet.

Cette archive devra contenir :

- les fichiers `.h` et `.c` (avec les 2 mises en œuvre de `matrice`), correctement commentés (chaque fonction/procédure doit avoir un commentaire, les invariants de boucle doivent être marqués), indentés, et codés (les noms de variables explicites, évitez les trop longues fonctions/procédures).
- le fichier `Makefile`, permettant de tester les 2 mises en œuvre de `matrice`) ;
- les fichiers de test de matrices et de vecteurs que vous avez utilisés ;
- un fichier `Documentation`, exclusivement au formet pdf, décrivant le fonctionnement général du programme et les originalités du code ;
- la compilation avec les options `-Wall -pedantic` ne doit pas donner de *warning*.

Attention **de bien respecter toutes** les directives indiquées dans ce projet. Si tel n'est pas le cas, la note s'en ressentira. Commencez à travailler ce projet dès à présent !

Have fun.