

Résolution d'un système linéaire à n équations à n inconnues

Igor APARICI - David ROUSSE

Résolution d'un système linéaire à n équations à n inconnues

<i>I – Présentation</i>	3
<i>II – Conventions de programmation</i>	4
<i>III – Gestion du projet</i>	5
a) Structure du projet	5
b) Fonctions utilisées	5
<i>IV – Méthode du pivot total</i>	7
<i>V – Méthode du pivot partiel</i>	7
<i>VI – Méthode du pivot de Gauss-Jordan</i>	7
<i>VII – Calcul du déterminant et du rang</i>	8
<i>VIII – Calcul de la matrice inverse</i>	8
<i>IX – Exemples des jeux d'essais</i>	8
<i>X – Conclusion</i>	10

I – Présentation

Soit $AX = B$ un système linéaire de n équations à n inconnues. Le programme gauss.exe se propose de déterminer si ce système admet des solutions et dans l'affirmative quelle est la solution de ce système.

Les méthodes de Gauss utilisées consistent en la transformation du système en un système équivalent ayant une forme particulière pour lequel la résolution est immédiate.

L'utilitaire réalisé propose l'ensemble des fonctionnalités suivantes pour la résolution du système linéaire carré :

```

      MENU DES FONCTIONS DE TRAITEMENT DU SYSTEME

*****
* 0. Quitter                                     *
* 1. Methode du pivot total                       *
* 2. Methode du pivot de Gauss-Jordan             *
* 3. Methode du pivot partiel                     *
* 4. Comparaison des resultats                    *
* 5. Calcul du determinant et du rang             *
* 6. Calcul de la matrice inverse                 *
* 7. Affichage des donnees                       *
* 8. Modifier un coefficient du systeme           *
* 9. Ecriture des resultats dans un fichier       *
* 10. Activation de l'ecriture sur disque         *
*****

Votre choix: _
```

Les fonctions utilisées pour l'implémentation de l'interface utilisateur présentée ci-dessus sont contenues dans le fichier gauss.c .

II – Conventions de programmation

Le programme sera écrit en C ANSI sous Borland C++ 5.00.

On utilisera un cartouche pour chaque module qui décrira sommairement les fonctionnalités contenues. Il comprendra donc la description et le but du module, la portée du module, les entrées et sorties nécessaires pour les modules qui utilisent des paramètres.

Il faudra, de plus, documenter à bon escient les lignes de codes, notamment pour la description des variables et des fonctions utilisées. Les commentaires seront marqués par `/* ...commentaires... */`

Par ailleurs, le fichier nommé **suivi.doc** contiendra un descriptif des différents modules du programme et de leur utilité.

Les noms des différents fichiers du projet seront enregistrés avec des noms courts (8 caractères maximum+extension) en minuscules.

Pour ce qui est de l'application, on choisira une projet 32 bits en mode console. Le développement en 32 bits permettra d'utiliser le débbugger sous Borland.

Enfin, on codera les données en double, c'est à dire sur 64 bits (si le système d'exploitation est 32 bits) et l'allocation dynamique de mémoire sera utilisée pour optimiser l'occupation de la RAM.

Remarque : pour information, voici les différents types de données utilisables sous Borland C++ 32 bits.

Type	Longueur	Plage
unsigned char	8 bits	0 à 255
char	8 bits	-128 à 127
short int	16 bits	-32,768 à 32,767
unsigned int	32 bits	0 à 4,294,967,295
int	32 bits	-2,147,483,648 à 2,147,483,647
unsigned long	32 bits	0 à 4,294,967,295
enum	16 bits	-2,147,483,648 à 2,147,483,647
long	32 bits	-2,147,483,648 à 2,147,483,647
float	32 bits	3.4×10^{-38} à $3.4 \times 10^{+38}$
double	64 bits	1.7×10^{-308} à $1.7 \times 10^{+308}$
long double	80 bits	3.4×10^{-4932} à $1.1 \times 10^{+4932}$
near (pointer)	32 bits	non définie
far (pointer)	32 bits	non définie

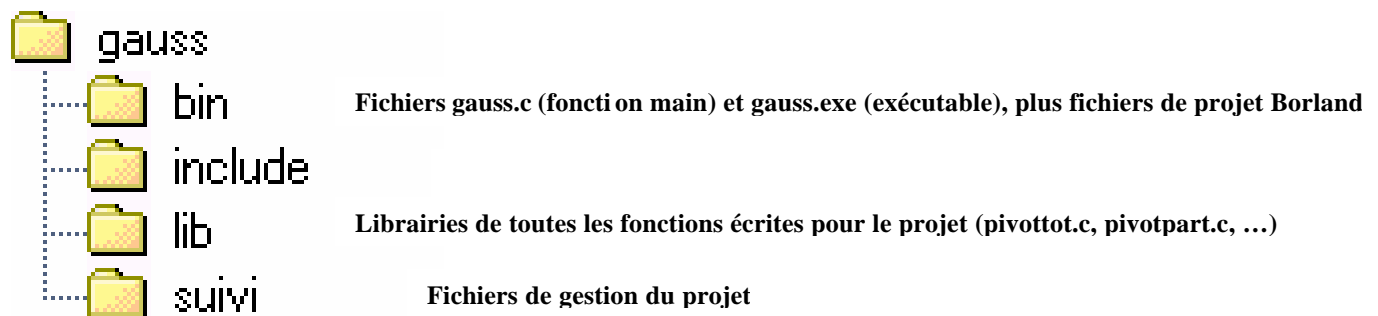
Une système à 100 équations à 100 inconnues occupera donc en mémoire, lors de l'exécution, 79 Ko (1 Ko = 1024 octets).

Les fonctions utilisées pour l'allocation dynamique de mémoire sont contenues dans le fichier mem.c.

III – Gestion du projet

a) Structure du projet

L'arborescence du projet est la suivante :



Les fichiers utilisés dans le projet sont les suivants :

```
gauss.c [.c] code size=2816 lines=433 data size=2844
..\lib\inverse.c [.c] code size=968 lines=196 data size=0
..\lib\pivottot.c [.c] code size=912 lines=187 data size=0
..\lib\detrang.c [.c] code size=480 lines=158 data size=0
..\lib\pivotgauss.c [.c] code size=636 lines=161 data size=0
..\lib\pivotpart.c [.c] code size=596 lines=170 data size=0
..\lib\utils.c [.c] code size=410 lines=134 data size=1364
..\lib\mem.c [.c] code size=303 lines=172 data size=132
..\lib\lecture.c [.c] code size=756 lines=176 data size=644
..\lib\ecriture.c [.c] code size=632 lines=205 data size=264
..\include\perso.h [.h]
```

b) Fonctions utilisées

Nom de la fonction	Description	Nom du fichier	Répertoire
main	exécutée au démarrage du programme	gauss.c	\bin
attendre	marque une pause dans l'exécution du programme	utils.c	\lib
menu	affiche le menu de choix des fonctionnalités du programme	utils.c	\lib
messagedebut	affiche un message de bienvenue	utils.c	\lib
messagefin	affiche un message de fin	utils.c	\lib
erreur	affiche un message d'erreur et arrête l'exécution du programme	utils.c	\lib

lecturetaille	lit dans un fichier la taille du système linéaire à résoudre	lecture.c	\lib
lecturedonnees	lit dans un fichier le système linéaire à résoudre	lecture.c	\lib
modifierdonnee	modifie un coefficient de la matrice en cours de traitement	lecture.c	\lib
affichervect	affiche un tableau	ecriture.c	\lib
affichermat	affiche une matrice	ecriture.c	\lib
ecriremat	ecrire une matrice dans un fichier	ecriture.c	\lib
ecrivevect	ecrire un vecteur dans un fichier	ecriture.c	\lib
ecriremessage	ecrire un message dans un fichier	ecriture.c	\lib
ecreireentier	ecrire un entier dans un fichier	ecriture.c	\lib
ecrirereel	ecrire un réel dans un fichier	ecriture.c	\lib
recuperernomfic	récupère un handle de fichier pour écriture	ecriture.c	\lib
allouervect	alloue de la mémoire pour un tableau	mem.c	\lib
allouermat	alloue de la mémoire pour une matrice	mem.c	\lib
allouer	alloue de la mémoire pour un système	mem.c	\lib
liberervect	libère la mémoire pour un vecteur	mem.c	\lib
liberermat	libère la mémoire pour une matrice	mem.c	\lib
occupmemvect	calcule la taille occupée en mémoire par un vecteur	mem.c	\lib
occupmemmat	calcule la taille occupée en mémoire par une matrice	mem.c	\lib
pivottot	résout le système par la méthode du pivot total	pivottot.c	\lib
pivotgauss	résout le système par la méthode du pivot de Jordan-Gauss	pivogauss.c	\lib
pivotpart	résout le système par la méthode du pivot partiel	pivotpart.c	\lib
detrang	calcul le rang et le déterminant du système	detrang.c	\lib
inverse	calcule la matrice inverse par la méthode du pivot total	inverse.c	\lib
-	ensemble des prototypes de toutes les fonctions du projet	perso.h	\include

IV – Méthode du pivot total

Le principe sommaire du pivot total est le suivant :

- Affectation d'une première valeur non nulle à pivotmax avec parcours de la matrice
- Recherche d'un premier pivot dans les parties non sélectionnées auparavant, afin de pouvoir effectuer des comparaisons avec un autre pivot dont la valeur absolue sera éventuellement supérieure à pivotmax
- Quand on trouve un pivot, on enregistre les lignes et les colonnes du pivot choisi
- La variable systemeindicateur est remplie de zéros dans les lignes et les colonnes correspondant à celles du pivot, dont les indices viennent juste d'être enregistrés
- Normalisation de la ligne du pivot.
- Réduction de la colonne du pivot
- S'il existe une solution au système, alors on ordonne le système en une matrice identité et on calcule la solution exacte

La fonction pivottot, codée dans le fichier pivottot.c, met en œuvre l'algorithme décrit ci-dessus.

V – Méthode du pivot partiel

Le principe du pivot partiel consiste à éliminer de proche en proche les coefficients sous-diagonaux en commençant par ceux de la première colonne et en choisissant à chaque étape un pivot non nul.

Quand on arrive à la k-ième colonne deux cas peuvent se présenter :

- Tous les coefficients $A[i][k]$ pour i de k à n sont nuls. Le système est alors non cramérien car en développant suivant les $k-1$ premières colonnes son déterminant est nul. On peut donc arrêter le traitement.
- L'un au moins des coefficients $A[i][k]$ pour i de k à n est non nul. On choisit alors $A[\text{ligne}][k]$ tel que ligne soit minimal. Si $\text{ligne} > k$, on échange les lignes d'indice ligne et k puis on réduit sous le pivot en remarquant que, étant donné le choix du pivot, les coefficients $A[k+1][k], \dots, A[\text{ligne}-1][k]$ sont déjà nuls, de même que $A[\text{ligne}][k]$: on réduit donc les lignes de ligne+1 à n avec la formule :

$$\text{ligne}[i] \leftarrow \text{systeme}[\text{col}][\text{col}] * \text{ligne}[i] - \text{systeme}[\text{ligne}][\text{col}] * \text{ligne}[\text{col}]$$

et on évite ainsi de diviser par un pivot proche de 0.

On peut remarquer que lorsqu'on arrive à la k-ième étape, les coefficients $A[i][j]$ tels que $k \leq i \leq n$ et $1 \leq j \leq k-1$ sont nuls donc pour faire une opération élémentaire sur 2 lignes appartenant à $[k, n]$, il suffit seulement d'opérer sur les indices de colonnes appartenant à $[k, n]$ ce qui permet d'échanger plus rapidement les lignes d'indices ligne et k .

La fonction pivotpart, codée dans le fichier pivotpart.c, met en œuvre l'algorithme décrit ci-dessus.

VI – Méthode du pivot de Gauss-Jordan

Une des variantes du pivot partiel consiste à normaliser la ligne du pivot avant la réduction. On appelle cette méthode la méthode de Gauss-Jordan.

Pour normaliser la ligne du pivot, on utilise la formule suivante :

$$\text{ligne}[i] \leftarrow \text{ligne}[i] / \text{systeme}[\text{colonnepivot}][\text{colonnepivot}]$$

Notons alors que lorsque la k-ième colonne a été traitée, elle devient stationnaire. De plus à la fin de la triangulation on trouve sur la diagonale les pivots successifs.

Remarque : il existe plusieurs variantes de cette algorithmes suivant la méthode choisie pour la recherche du pivot. Notamment, on aurait pu choisir un pivot de module maximal colonne par colonne.

La fonction pivotgauss, codée dans le fichier pivotgauss.c, met en œuvre l'algorithme décrit ci-dessus.

VII – Calcul du déterminant et du rang

On utilise ici l'algorithme du pivot partiel en laissant de côté toutes les manipulations concernant le second membre et le calcul des solutions.

De plus, on peut s'arranger pour que les opérations effectuées sur la matrice ne modifie pas son déterminant. Pour faire cela, lors du traitement de la k-ième colonne, on amène le pivot sur la diagonale par $\text{ligne}[k] \leftarrow \text{ligne}[k] + \text{ligne}[\text{ligne}]$ puis on élimine les coefficients sous le pivot.

Ainsi, on ne fait que des opérations de type addition d'un multiple d'une ligne à une autre ligne, qui ne modifie pas le déterminant du système.

Le déterminant est nul si à la k-ième étape on ne trouve pas de pivot non nul, ou bien est le produit des pivots.

En ce qui concerne le calcul du rang du système, il correspond au nombre d'opérations nécessaires pour trianguler le système.

La fonction detrang, codée dans le fichier detrang.c, met en œuvre l'algorithme décrit ci-dessus.

VIII – Calcul de la matrice inverse

On utilise l'algorithme du pivot total pour calculer la matrice inverse du système.

On pourrait faire appel à n fonctions pivottot en prenant comme second membre les colonnes de la matrice unité. Cela est toutefois assez coûteux car les opérations effectuées sur le premier membre lors de la triangulation seraient effectuées n fois. On peut, à la place, appeler une seule fois la fonction pivottot avec comme second membre la matrice unité.

En procédant ainsi la matrice du premier membre est transformée de proche en proche en la matrice unité et la matrice unité en la matrice inverse du système.

La fonction inverse, codée dans le fichier inverse.c, met en œuvre l'algorithme décrit ci-dessus.

IX – Exemples des jeux d'essais

La possibilité que donne gauss.exe de lire et d'écrire les données à partir de fichiers disque donne une grande facilité et confort d'utilisation. En effet, cela évite les problèmes de saisie, permet le test de système de grande taille et donne la possibilité de conserver les résultats durablement.

Au lancement du programme les données du système sont lues dans un fichier texte :

```
Bienvenue dans le programme Gauss ...
Cet utilitaire vous permet de realiser diverses operations
sur des systemes lineaires a n equations a n inconnues.

Appuyez sur ENTREE pour continuer ...

Entrez le nom du fichier (nom et extension).
Note: si aucun nom de fichier entre, le programme essaiera
de lire donnees.txt dans le repertoire courant.
Nom fichier: _
```


En ce qui concerne les résultats, on peut choisir l’affichage sur l’écran et/ou sur le disque :

```
Resolution du systeme par la methode du PIVOT TOTAL ...  
Solution du systeme :  
  
    0.750000 0.500000 -0.250000  
  
Ecriture des resultats sur le disque ...  
  
Entrez le nom du fichier (nom et extension).  
Note: si aucun nom de fichier entre, le programme essaiera  
d'ecrire dans max.txt dans le repertoire courant.  
    Nom fichier: _
```

L'utilitaire gauss.exe génère les fichiers résultats par défaut suivants :

- sortie.txt : fichier de sortie par défaut (choix indice 9)
- max.txt : fichier après utilisation de la méthode du pivot total (choix indice 1)
- gauss.txt : fichier après utilisation de la méthode de Gauss-Jordan (choix indice 2)
- partiel.txt : fichier après utilisation de la méthode du pivot partiel (choix indice 3)
- compare.txt : fichier après utilisation des 3 méthodes (choix indice 4)
- detrang.txt : fichier après calcul du déterminant et du rang (choix indice 5)
- inverse.txt : fichier pour calcul de la matrice inverse via la méthode du pivot total (choix indice 6)

Les fonctions utilisées pour la lecture des données sur disque sont contenues dans le fichier lecture.c.

Les fonctions utilisées pour l’écriture des données à l’écran et/ou sur disque sont contenues dans le fichier ecriture.c.

Pour ce qui est des jeux d'essais, la résolution de systèmes courants ne pose aucun problème et les résultats par les trois méthodes sont identiques.

Les systèmes mal conditionnés sont également résolus avec une excellente précision.

Système contenu dans le fichier malcond1.txt :

$$\begin{cases} 0.1000 & 1 & = & 1 \\ 1 & 1 & = & 2 \end{cases}$$

Solution exacte	Pivot total	Gauss-Jordan	Pivot partiel
1.00010	1.111111	1.111111	1.111111
0.99990	0.888889	0.888889	0.888889

Système contenu dans le fichier malcond2.txt :

$$\begin{cases} 21 & 130 & 0 & 2.0 & = & 153.1 \\ 13 & 80 & 4.7^8 & 752 & = & 849.74 \\ 0 & -0.4 & 3.9816^8 & 4.2 & = & 7.7816 \\ 0 & 0 & 1.7 & 9^8-9 & = & 2.6^8-8 \end{cases}$$

Solution exacte	Pivot total	Gauss-Jordan	Pivot partiel
0.1000000E+01	1.005716	1.005716	1.005716
0.1000000E+01	0.999846	0.999846	0.999846
0.9999999E-08	0.000000	0.000000	0.000000
0.1000000E+01	0.999971	0.999971	0.999971

Le listing du programme est fournit avec le présent rapport. A noter qu'il est intéressant de lire ce code source qui est commenté au maximum, dans un soucis d'évolutivité.

X – Conclusion

Le projet nous a permis de réfléchir sur la manière de représenter des nombres réels sur un micro ordinateur. Cette expérience enrichissante nous a également donné l'opportunité d'appliquer nos connaissances informatiques aux mathématiques. Néanmoins le langage C n'est peut être pas le plus approprié aux calculs numériques en binaire : Lisp ou Maple, par exemple, auraient été mieux adaptés.