DIF Clément
LOXOL Nicolas

Compte rendu TP : Résolution de systèmes linéaires

1. Rappel des méthodes

Les quatre méthodes qui vont être présentées permettent la résolution de systèmes linéaires à n équations et n inconnues. Un système de telle sorte se présente sous forme d'un ensemble d'équations représentées toutefois dans des matrices pour simplifier la résolution.

Un système linéaire classique de cette catégorie s'écrit AX = B ou A (d'ordre n) représente la matrice (contenant les coefficients des équations) ; B est le second membre (matrice colonne), c'est-à-dire les résultats aux équations de A et X, le vecteur colonne solution que l'on souhaite déterminer.

Il existe deux types de méthodes de résolution :

- ✓ **Les méthodes directes** : Transformer le système de façon à obtenir des équations plus faciles et plus rapides à résoudre. La solution obtenue est exacte.
- ✓ **Les méthodes indirectes ou itératives** : Choisir un vecteur initial puis passer, par un certain processus, d'un vecteur à un autre jusqu'à atteindre l'un de ceux qui se rapprochent de la solution.

Méthodes directes implémentées :

Méthode de Gauss : Cette méthode permet de résoudre les systèmes linéaires grâce à des opérations élémentaires sur les lignes qui permettent d'échelonner la matrice c'est-à-dire la rendre triangulaire inférieure. Une fois la matrice (A|B) échelonnée ; il suffit de résoudre le système d'équations équivalent alors très simple qui permet de trouver les inconnues du vecteur colonne X.

Méthode de Cholesky : Cette méthode décompose la matrice A en un produit de deux matrices R^tR. R est alors une matrice triangulaire supérieure et R^t sa transposée. On a donc $R^tRX = B$. Par la suite, on résout d'abord $R^tY = B$ où $Y = RX$ afin de déterminer le vecteur colonne Y. Puis on calcule $RX = Y$. Cela nous permet de déterminer les composantes de X.

Remarque : La méthode de Cholesky s'applique à une matrice symétrique ($A^t = A$) et définie positive ($xAx^t > 0$ pour tout x non nul).

Méthodes indirectes :

Méthode de Jacobi : La décomposition de A se fait sous la forme D-E-F ou D est la matrice diagonale formée des éléments de la diagonale de A, -E est la matrice triangulaire inférieure composée des éléments de A tandis que -F est la matrice triangulaire supérieure formée des éléments de A. On pose A = M – N avec M = D et N = E + F. En remplaçant on a : $AX = B \Leftrightarrow MX = NX + B \Leftrightarrow M^{-1}NX + M^{-1}B = F(x)$ ou F est une fonction affine avec point fixe.

Remarque : Si A est une matrice d'ordre n définie positive ($xAx^t > 0$ pour tout x non nul) et à diagonale strictement dominante (voir formule ci-dessous), la méthode de Jacobi converge.

$$\forall i = 1, \dots, n \left|a_{i,i}\right| > \sum_{\substack{j=1 \\ j \neq i}}^{n} (|a_{i,j}|)$$

Méthode de Gauss-Seidel : On résout de la même manière que dans l'algorithme de Jacobi à une différence près : ici M = D - E et N = F.

Remarque : Si A est une matrice définie positive ($xAx^t > 0$ pour tout x non nul) et est symétrique alors la méthode de Gauss-Seidel converge. L'on peut également noter que la méthode de Gauss-Seidel converge plus rapidement que celle de Jacobi.

2. Présentation du programme

Notre programme se sépare en plusieurs parties distinctes :

- Les « includes » qui permettent d'utiliser des fonctions présentes dans la bibliothèque standard (par exemple les fonctions mathématiques, les fonctions concernant l'allocation dynamique, la génération aléatoire de nombres, etc.).

- Les « defines » où deux constantes sont définies (TRUE = 1 et FALSE = 0), elles correspondent aux booléens qui ne sont pas définis en C avant le C99.

- La création d'une structure optimisation contenant deux variables : la première représentera le nombre de clocks ; un clock étant une unité de temps processeur (valant généralement, par convention, une micro seconde) tandis que la seconde contiendra le nombre d'octets alloués.

- Un typedef qui permet l'introduction d'une macro « opt » à la place de « struct optimisation ».

- Les prototypes contenant :

  - Des fonctions diverses conçues pour les allocations dynamiques de mémoire ou l'affichage ainsi qu'une fonction min pour trouver le minimum entre deux entiers.
  - Les fonctions de résolution utilisant chacune des méthodes spécifiée ci-dessus.
  - Des fonctions annexes nécessaires au bon fonctionnement de la méthode Cholesky.

- Une fonction qui permet de tester l'optimisation des différentes méthodes adjointe à d'autres testant si la matrice remplit les critères de symétrie et de diagonale dominante.
- Les fonctions de génération de matrices types.

- La fonction *main* (point d'entrée du programme) : Nous créons une matrice (représentant A) en utilisant une des fonctions de génération de matrice puis définissons le vecteur X contenant les inconnues et enfin, nous écrivons le second membre. Nous affichons ensuite, pour chaque méthode, le vecteur colonne solution X trouvé ainsi que le degré optimisation obtenu via la fonction citée ci-dessus.

- *min* → calcul le plus petit entier entre a et b fournis en paramètres.

- *allouer_memoire_matrice* → alloue dynamiquement une matrice de « *doubles* », teste si l'allocation a fonctionné et retourne la matrice allouée.

- *liberer_memoire_matrice* → libère la mémoire utilisée par la matrice.

- *afficher_matrice_et_second_membre* → affiche la matrice et son second membre.

- *afficher_solutions_chaque_methode* → affiche les solutions trouvées par les différentes méthodes (respectivement Gauss, Cholesky, Jacobi, et Gauss-Seidel).

- *gauss* (respectivement *cholesky*, *jacobi*, *gauss_seidel*) → résout le système AX=B selon la méthode éponyme.

- *trouver_decomposition* → permet de trouver la décomposition $A=R^tR$ nécessaire dans la méthode de Cholesky.

- *resyst_tri_inf* et *resyst_tri_sup* → permettent de résoudre respectivement $R^tY = B$ et $RX = Y$ dans la méthode de Cholesky.

- *tester_optimisation* -> Affiche les clocks nécessaire et les octets alloués pour chaque méthode de résolution.

- Les autres fonctions permettent de créer les matrices test.

Remarques :

- « *int n* » présent dans certains prototypes représente l'ordre de la matrice carrée.

- La fonction *tan* présente dans *generer_matrice_creuse* permet de générer des nombres aléatoires en utilisant le domaine de définition de la fonction tangente sur $[-\frac{\pi}{2} ; \frac{\pi}{2}]$. Les nombres aléatoires fournis se rapprochent de 0 permettant ainsi, de manipuler des grandeurs pratiques (ne dépassant généralement pas 10).

- Le fichier tp2.c offre des commentaires supplémentaires si nécessaire.

3. Jeux d'essais

Nous avons choisi d'utiliser les matrices de test. Pour les méthodes directes, nous avons fait varier la taille des matrices (*n*) tandis que pour les méthodes itératives, nous nous sommes occupés de changer, en plus de la taille des matrices, la précision (*e*). Le nombre d'itérations, quant à lui, est resté fixé à $2n^2$.

Nous avons également pris soin d'observer la vitesse de convergence pour chaque méthode.

4. Commentaires sur les jeux d'essais

La précision n'influe pas sur le nombre d'octets alloués mais bien sur le temps d'exécution. Les méthodes les moins coûteuses sont Gauss, Gauss-Seidel et Cholesky. Cependant, la mémoire allouée par Cholesky est trois fois plus élevée que celle allouée par les autres méthodes. Jacobi reste enfin la moins optimale. On remarque néanmoins que la méthode la plus appropriée sur une précision extrême est celle de Gauss. C'est donc une méthode directe qui est la plus efficace.

Précision ->	0.1	0.01	0.001	1^{E-18}
Gauss	1 clock 56 octets	1 clock 56 octets	1 clock 56 octets	1 clock 56 octets
Cholesky	3 clocks 144 octets	2 clocks 144 octets	2 clocks 144 octets	3 clocks 144 octets
Jacobi	44 clocks 52 octets	22 clocks 52 octets	25 clocks 52 octets	110 clocks 52 octets
Gauss-Seidel	3 clocks 52 octets	3 clocks 52 octets	3 clocks 52 octets	4 clocks 52 octets
Méthode(s) la/les plus efficace(s)	Gauss Gauss-Seidel	Gauss Gauss-Seidel	Gauss Gauss-Seidel	Gauss

La taille de la matrice influe directement sur le temps de résolution mais également sur le nombre d'octets alloués. Effectivement, pour une taille de matrice inférieure à 4, la méthode de Gauss-Seidel reste la plus efficace. Cependant, lorsque la taille de la matrice atteint 6, la méthode de Gauss devient alors la plus efficiente. En outre, pour une taille de matrice très élevée, la méthode de Gauss demeure optimale sur la mémoire allouée même si celle de Gauss-Seidel demeure la plus rapide.

Taille ->	2	3	4	6	100
Gauss	1 clock 56 octets	1 clock 56 octets	2 clocks 56 octets	2 clocks 56 octets	2386 clocks 56 octets
Cholesky	2 clocks 104 octets	3 clocks 144 octets	3 clocks 144 octets	17 clocks 184 octets	863 clocks 4024 octets
Jacobi	18 clocks 44 octets	21 clocks 52 octets	200 clocks 60 octets	266 clocks 76 octets	42759 clocks 828 octets
Gauss-Seidel	2 clocks 44 octets	2 clocks 52 octets	4 clocks 60 octets	5 clocks 76 octets	647 clocks 828 octets
Efficacité Temps	Gauss Cholesky Gauss-Seidel	Gauss Cholesky Gauss-Seidel	Gauss Cholesky Gauss-Seidel	Gauss Gauss-Seidel	Gauss-Seidel Cholesky
Efficacité mémoire	Jacobi Gauss-Seidel	Jacobi Gauss-Seidel	Gauss Jacobi Gauss-Seidel	Gauss Jacobi Gauss-Seidel	Gauss

Nous avons pu également remarquer, grâce au compteur d'itérations, que la méthode de Gauss-Seidel converge plus vite que la méthode de Jacobi. Cela est d'autant plus vrai si on augmente la taille de la matrice ou la précision.

La complexité théorique des méthodes Gauss et Cholesky s'élève à O(n3). Tandis que celle des méthodes Jacobi et Gauss-Seidel s'exhausse seulement à $O(n^2)$.

De plus, dans le cas des méthodes itératives, on remarque une accumulation d'erreurs dues aux approximations qui rallonge alors, le temps de convergence.

5-Conclusion :

D'après les tests précédents ; la méthode de Gauss semble la plus adaptée pour une précision élevée et alloue peu de mémoire proportionnellement à l'augmentation de l'ordre de la matrice.

Néanmoins, la méthode de Gauss-Seidel domine nettement en temps de résolution lorsque la taille de la matrice augmente.

Il revient donc à l'utilisateur le choix de choisir sa méthode en fonction de ses attentes en temps d'exécution et en espace de stockage.