Compte Rendu Projet en C

Xavier Krsinar - 26102387

Introduction

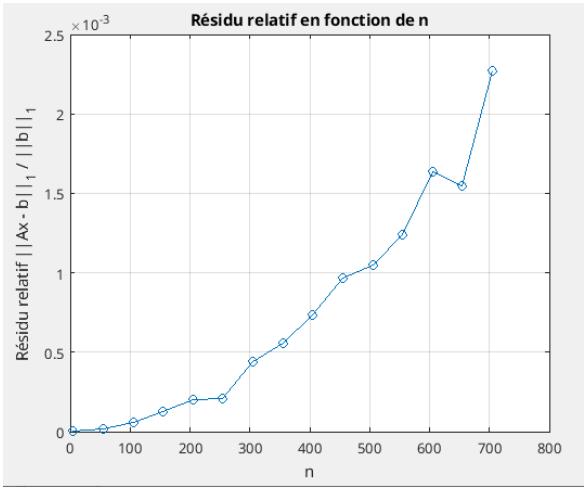
Dans ce projet, on résout une équation différentielle ordinaire $-u'' = f$ sur l'intervalle [0,1]. On utilise la méthode des différences finies pour la discrétisation avec un pas $h = 1/(n+1)$. Cela ramène donc à la résolution d'un système $Ax = b$ de dimension n.

Pour exécuter le programme, on utilise la commande `bash projet.sh` (ou `projetmono.sh`), qui crée un exécutable `projet.exe` (ou `projetmono.exe`).

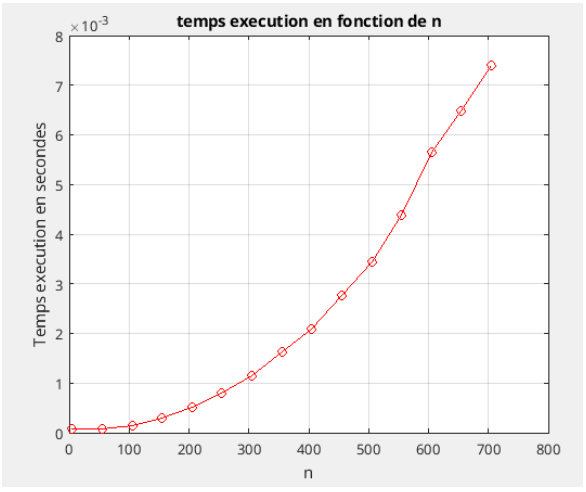
On utilise la fonction `LAPACKE_sgesv` pour résoudre le problème. Cette fonction est l'interface C de la bibliothèque LAPACK, qui est nativement en Fortran. Le sujet indique que Fortran stocke les matrices en *column major* (colonne par colonne). Cependant, nous avons utilisé `LAPACK_ROW_MAJOR` lors de l'appel de la fonction, ce qui indique à LAPACKE de faire la transition attendue vers des matrices stockées en *column major*. Ici, cela revient au même car la matrice A est symétrique, mais ça peut être utile pour d'autres matrices.

La fonction prend en argument des pointeurs vers les matrices A et b. Elle va écraser ces dernières avec la décomposition LU pour la matrice A et la solution x pour la matrice b. Pour calculer le résidu il faut donc dupliquer ces matrices avant l'appel de la fonction (Aprim et Bprim dans le code).

Les courbes ci-dessous ont été créées avec MatLab selon les données générées par le programme.



(a) Résidu relatif en fonction de n



(b) Temps d'exécution en fonction de n

FIGURE 1 – Résultats numériques en fonction de n

Conclusion de l'observation des graphiques

Le graphique ci-dessus montre que la norme relative du résidu augmente progressivement en même temps que n, la taille du système. La norme du résidu relatif commence de l'ordre de 10^{-7}, soit proche de l'epsilon machine en simple précision (23 bits dans la mantisse, précision

$\approx 2^{-23} \approx 1.19 \times 10^{-7}$). Même si elle reste faible pour un grand n, elle reste significative. Cela met en avant la limite de la simple précision. De plus, le conditionnement de la matrice de discrétisation A est mauvais. Son conditionnement évolue de manière quadratique ; plus elle augmente en taille, plus le conditionnement est grand. Ainsi, le couplage de la simple précision avec un conditionnement en $O(n^2)$ explique la croissance de l'erreur sur le graphique.

Par rapport au temps d'exécution, la courbe croît très rapidement. Cela est dû au fait que la décomposition LU classique a une complexité algorithmique en $O(\frac{2}{3}n^3)$. Ainsi, quand n augmente, le temps "explose".

Mauvaise mise en œuvre

La mise en œuvre n'est pas judicieuse. On utilise `LAPACKE_sgesv()`, qui est conçue pour résoudre des systèmes linéaires $Ax = b$ avec A une matrice dense. La complexité algorithmique est en $O(\frac{2}{3}n^3)$. Or la matrice A de notre problème est tridiagonale, et il existe des algorithmes de résolution des matrices tridiagonales bien moins coûteux que ce que nous utilisons. On fait donc de nombreuses opérations inutiles sur les 0 de la matrice. Le graphique du temps d'exécution l'illustre bien.

De plus, on stocke n^2 éléments, ce qui fait environ 500 000 valeurs pour $n = 705$. Cela pourrait être fortement réduit car, en utilisant un algorithme qui exploite le caractère tridiagonal, on pourrait se limiter à $3n - 2$ valeurs, soit 2113 pour $n = 705$. On utilise presque toute la mémoire pour ne rien stocker.

Améliorations possibles

Une première amélioration serait d'écrire le programme en double précision. On ferait passer la précision machine à $\approx 2.22 \times 10^{-16}$. Il faudrait donc adapter tout le programme, en utilisant `LAPACK_dgesv()`, changer les "float" avec des "double", ...Cela permettra aussi de lutter contre l'amplification des erreurs due au mauvais conditionnement de la matrice.

Nous allouons une matrice dense $n \times n$, presque remplie de 0. Pour optimiser cela, nous pourrions stocker une matrice à 3 colonnes ; une pour la sous-diagonale, une pour la diagonale et une pour la sur-diagonale.

Avec cette nouvelle façon de stocker la matrice A, on pourrait utiliser un algorithme comme l'algorithme de Thomas vu précédemment, qui est plus efficace au vu de la structure. Il permet une utilisation limitée de la mémoire. Cet algorithme est même implémenté dans la fonction `LAPACK_sgtsv`, qui est donc faite pour les matrices tridiagonales en simple précision. Cette fonction prendra en argument une matrice A tridiagonale sous forme de trois tableaux distincts et résout $Ax = b$. Pour aller plus loin, nous pourrions même utiliser `LAPACK_sptsv()`, qui utilise le critère symétrique définie positive de la matrice, qui utilise une factorisation $A = LDL^T$, ce qui réduirait encore le temps de résolution.

La complexité passerait donc de $O(n^3)$ à $O(n)$ et donc le temps de calcul serait significativement plus rapide. De plus, on ne stockera plus que 3 vecteurs au lieu d'une matrice carrée, ce qui réduira l'occupation en mémoire de n^2 à $3n$.

On en conclut donc que même si la méthode est censée converger, l'erreur numérique domine à cause notamment de la simple précision et du mauvais conditionnement de la matrice. De plus, ce modèle pourrait poser des problèmes avec un n très grand, dans d'autres usages, causant des erreurs telles qu'un dépassement de mémoire. Ce problème serait résolvable avec un algorithme adapté.