Rapport de TP Calcul Numérique

Nsundi Mboli Patrick <patrick.nsundi-mboli@ens.uvsq.fr>

Tutuors:
M.DUFAUD Thomas <thomas.dufaud@uvsq.fr>
M.GURHEM Jérome <jgurhem@aneo.fr>

05/01/2025

Lien du dépot Github du projet (pour la partie 5) : `https://github.com/ElitePat/tp_poisson`

Table des matières

Première partie

Implémentation en Scilab

1 TP 2 : (Mise en place de l'environment Scilab)

Les fichiers mentionés dans cette section se trouvent dans le sous-repertoire TP_2.

1.1 Exercice 7 : Matrice random et problème "jouet" :

Pour repondre à cette question nous avons écrit dans le fichier "exo7_tp2.sci" un script Scilab qui stocke dans un vecteur RES les erreurs avant et arrière de la resolution du système $Ax = b$ (avec $A \in \mathbb{R}^{n\times n}$,$b, x \in \mathbb{R}^n$). Le script fait appel a deux autres fichiers pour cela : "erreur_avant.sci" et "erreur_arrière.sci". Les resultats obtenus pour les differentes taille de matrices sont dans le tableau ci-dessous :

Taille Matrice	Erreur	
	Erreur avant	Erreur arrière
3	$2.41e-16$	0
10	$2.61e-16$	0
100	$4.38e-14$	0
1000	$6.80e-13$	0
10000	$1.72e-10$	0

TABLE 1 – Évolution de l'erreur arrière et avant avec differentes tailles de système linéaires

D'aprés le graphique on voit que l'augmentation de la taille des matrices fait seulement acroître l'erreur arrière, l'erreur avant restant nulle.
L'augmentation de l'erreur avant semble cohérente. ...

1.2 Exercice 8 : Produit Matrice-Matrice :

Le fichier "matmat3b.sci" contient le code de l'algorithme de multiplication matricielle avec 3 boucles.
Le fichier "matmat2b.sci" contient le code de l'algorithme de multiplication matricielle avec 2 boucles.
Le fichier "matmat1b.sci" contient le code de l'algorithme de multiplication matricielle avec 1 boucle.
Le fichier "exo8_test.sci" lui, permet d'automatiser les tests avec une foncion test()

qui prend en argument un entier l. La fonction retournera alors les temps d'executions de differents algorithmes de multiplication de matrices dans un format matriciel et pour differentes tailles de matrice, allant de 2 à 2^l. Ici nous avons testé la fonction avec $l = 8$, et voici les resultats :

Taille Matrice $(n * n)$	Algorithme écrit avec		
	3 Boucles	2 Boucles	1 Boucle
2	0.000152	0.0001669	0.0001482
4	0.000506	0.0001612	0.0001146
8	0.0025783	0.0005274	0.0001319
16	0.0194199	0.0019644	0.0002459
32	0.162283	0.0081325	0.0004988
64	1.2131904	0.035076	0.0010432
128	9.779465	0.1563281	0.002338
256	77.721955	0.7919257	0.0062288

TABLE 2 – Comparaison des 3 version de l'alogrithme de multiplication matricielle avec differentes tailles de matrice en temps d'execution

Et voici les graphiques asociés (faits sur Excel) :

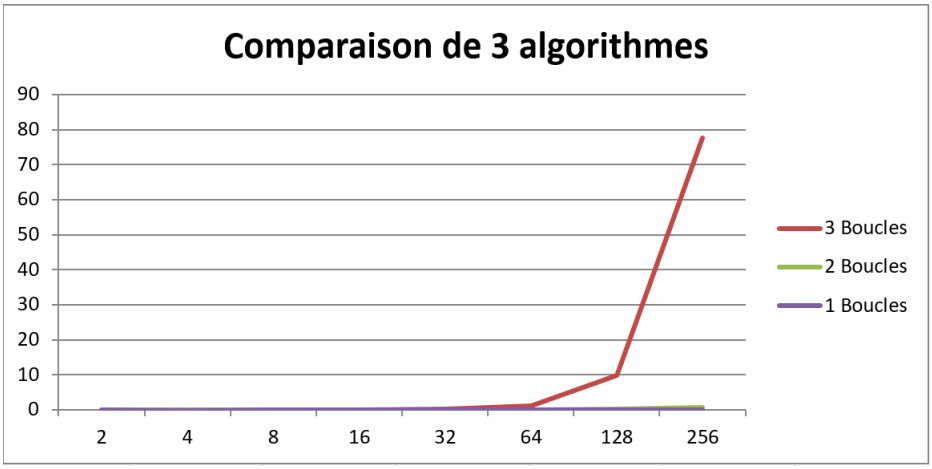

FIGURE 1 – Graphique associé au tableau 2

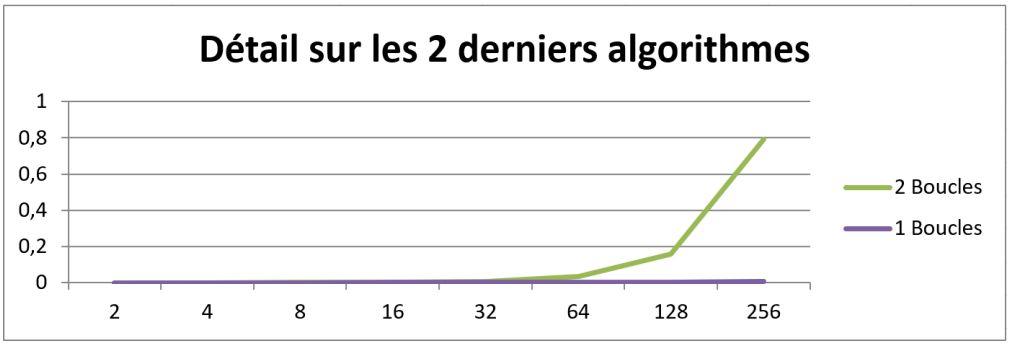

FIGURE 2 – Zoom pour les algorithmes à 2 et 1 boucles dans le graphe 1

Avec le graphique 1 on observe que la multiplication matricielle a 3 boucles subit un temps d'execution de plus en plus long avec des grandes matrices (à partir d'une taille de 100). Contrairement aux deux autres alogrithmes dont le temps d'execution semble constant. Mais avec le graphique 2, nous pouvons observer que le temps d'execution de l'algorithme a 2 boucles augmente aussi avec la taille de la matrice, bien que restant en dessous de la seconde.
Ces observations sont le resultat d'une vectorisation des calculs faite par Scilab. Soit au lieu d'aller chercher en mémoire les valeurs une par une, il les prendrait en groupe de vecteurs, et ferait une multiplication de vecteurs. Cela est d'autant plus efficace qu'il n'y a aucune dependence entre les données en lecture, ce qui permet de faire plusieurs opérations à la fois.

2 TP 3 : Résolution de système linéaire

2.1 Exercice 2 : Système triangulaire :

Le fichier "exo2_tp3.sci" permet de verifier la validité des fonctions "usolve(U,b)" et "lsolve(L,b)" respectivement implementés dans les fichiers "usolve.sci" et "lsolve.sci" en effectuant les calculs suivants :

2.2 Exercice 3 : Gauss :

Le fichier "exo2_tp3.sci" permet de verifier la validité de la fonction "gauss-kij3b(A,b)" en faisant les calculs suivants :

2.3 Exercice 4 : LU :

Le fichier "fact_LU.sci" contient le code de la fonction "mylu3b(A)". Le test et la validation de l'algorithme est faite dans le fichier "test_fact_LU.sci" de la manière suivante. A la fin on obtient un vecteur de matrices (soit une matrice) des erreurs de factorisation LU pour tous les cas.

Deuxième partie

Implémentation en C

3 TP 5 : Methodes itératives de base (suite)

L'objectif de cette partie du TP était d'implementer en C des fonctions qu'on a utilisés où écrites en Scilab, pour nous confronter à l'appel de bibliothèques de calcul BLAS et LAPACK.

A présent la complexité temporelle de chaque fonction sera presenté en rouge et la complexité spaciale en bleu.

3.1 Fonctions pour méthode directe de résolution :

- `set_GB_operator_colMajor_poisson1D()`

  $O(n^2)$ | $O(n^2)$ | Affectation matrice de Poisson 1D, pas d'erreurs de précision attendues.

- `set_GB_operator_colMajor_poisson1D_Id()`

  $O(n^2)$ | $O(n^2)$ | Affectation matrice identité, pas d'erreurs de précision attendues.

- `set_dense_RHS_DBC_1D()`

  $O(n)$ | $O(n)$ | Affectation d'un vecteur, pas d'erreurs de précision attendues.

- `set_analytical_solution_DBC_1D()`

  $O(n)$ | $O(n)$

- `set_grid_points_1D()`

  $O(n)$ | $O(n)$

- `relative_forward_error()`

  $O(n^2)$ | $O(n)$

- `indexABCol()`

  $O(1)$ | $O(1)$

3.2 Fonctions pour méthode itérative de résolution :

- `eig_poisson1D()`

  $O(n)$ | $O(n)$

- `eigmax_poisson1D()`

  $O(1)$ | $O(1)$
- `eigmin_poisson1D()`

  $O(1)$ | $O(1)$
- `richardson_alpha_opt()`

  $O(1)$ | $O(1)$
- `richardson_alpha()`

  Non pertinent pour une méthode itérative | $O(n^2)$
- `extract_MB_jacobi_tridiag()`

  $O(n^2)$ | $O(n^2)$
- `extract_MB_gauss_seidel_tridiag()`

  $O(n^2)$ | $O(n^2)$
- `richardson_MB()`

  Non pertinent pour une méthode itérative | $O(n^2)$

Remarque : Il faut souligner ici que nous traitons avec des matrice tridiagonales stockés en General Band. Ainsi, si on suppose que n c'est la taille de la diagonale et non la dimension de la matrice (matrice de taille $n \times n$ alors) ou du vecteur, les complexités en $O(n^2)$ devienent des complexités en $O(3n) \equiv O(n)$.