1 Introduction

Le calcul matriciel et la résolution de système linéaires ont toujours été un problème majeur pour le monde scientifique.C'est dans ce contexte qu'on va essayer de voir à travers nos travaux dirigés et pratiques comment on peut choisir ou écrire un bon algorithme capable de résoudre ce problème avec un temps record et avec une occupation en mémoire réduite.Pour cela on procède pas à pas pour étudier les méthodes des algorithmes suivants.

2 Exercice 6 Rappel de *scilab*

- Question 1.
  $x = rand(3, 1)$ est vecteur de taille 3 lignes et 1 colonne.

- Question 2.
  $y = rand(4, 1)$ est vecteur de taille 4 lignes et 1 colonne.

- Question 3.
  Les opérations $z = x + y$ et $s = xy$ ne peuvent pas se réaliser car la taille du vecteur x qui est 3 lignes et 1 colonne est différente de celle de y qui est 4 lignes et 1 colonne.
  Donc pour réaliser des opérations en matrices ou en vecteurs les tailles doivent etre égales.

- Question 4.
  Avec la fonction size(), on a calculé la taille de x qui est de 3 lignes et 1 colonne et celle de y qui est de 4 lignes et 1 colonne.

- Question 5.
  La fonction $norm()$ de *scliab* nous permet de calculer la norme de x avec par exemple $x = rand(1, 4)$ on a la norme de x = 1.25 et occupe en mémoire 216 bytes.

- Question 6.
  La matrice $A = rand(4, 3)$ est une matrice de 4 lignes et de 3 colonnes.

- question 7.
  A' est la transposée de la matrice A, donc elle est de 3 lignes et de 4 colonnes.

- Question 8.
  Soient $A = rand(4,4)$ et $B = rand(10,10)$.
  On effectue les opérations élémentaire des deux matrices A et B.
  on a :
  $det(A) = -0.174$ et occupe en mémoire 216 bytes;
  $det(B) = -0.0267$ et occupe aussi en mémoire 216 bytes;
  Pour $A = rand(10,10)$ et $B = rand(10,10)$ deux matrices carrées on peut effectuer ses opérations suivantes: $A + B =$ matrice carrées de taille 10;
  $A * B =$ matrices carrées de taille 10;

- question 9.
  On calcul le conditionnement de la matrice A.
  avec $A = rand(10,10)$
  $cond(A) = 37.8$ et occupe en mémoire 216 bytes;

3 Exercice 7 Matrice *random* et problème "jouet"

- Question 1.
  On écris une matrice A de taille $3 * 3$: $A = rand(3,3)$.

- question 2.
  On écrit un vecteur xex dans R^3 avec la fonction $rand()$: $xex = rand(3,3)$.
  on vérifie bien avec la fonction $size()$ que xex est un vecteur colonne car sa taille est : $3 * 1$.

- question 3.
  On écrit b le produit de $A * xex$: $b = A * xex$.
  Avec la fonction $size()$,on voit que b est vecteur colonne car $3 * 1$.

- question 4.
  Avec la fonction on résolve l'équation Ax = b avec la solution de scilab qui est x.
  L'équation admet une unique solution car le determinant de A est non null.

- question 5.
  Pour cet exemple on voit que l'erreur en avant est de l'ordre de $err = 1.19e^{-16}$.
  Donc les données entrées sont perturbées.

- question 6.
  Pour cet exemple aussi on voit que l'erreur en après est de l'ordre de $relres = 0$.
  Donc le vecteur b entré est égal au produit Ax.

- question 6.
  Pour $n = 100$
  On voit que l'erreur avant augmente car $err = 1.42e^{-13}$ et l'erreur après reste égale à $relres = 0$.
  Donc les données entrées sont perturbées.

  Pour $n = 1000$ On voit que l'erreur avant augmente encore de l'ordre de $err = 4.03e^{-12}$ et l'erreur après reste égale à $relres = 0$. Donc les données entrées restent perturbées.

  Pour $n = 1000$ On voit que l'erreur avant augmente de plus en plus et elle est de l'ordre $err = 1.86e^{-10}$ et l'erreur après est toujours égale à $relres = 0$

  Conclusion exo7
  On voit que l'erreur avant est proportionnelle à la taille de la matrice car pour n assez grand l'erreur avant augmente c'est à dire les données entrées deviennent de plus en plus perturbées. l'erreur est exponentielle.
  Donc pour mieux voir l'erreur on fait appel au conditionnement pour pouvoir évaluer pour une matrice A sa capacité de générer de une erreur.
  Et de plus en plus on voit utiliser de la mémoire donc si on continue on pourrait confronter à une segmentation de la mémoire.

4 Exercice 8 Produit Matrice-Matrice

* question 1.
  On a écrit l'algorithme du produit matrice-matrice dans un fichier *matmat3b.sci* avec trois boucles sous la forme ijk en créant une fonction
  $$function[C] = matmat3b(A, B)$$
  et on voit que il y a $m * n$ nombres multiplications et $(m * n - 1)$ nombres d'addition.
  Donc cet algorithme est de niveau 3 c'est à dire de l'ordre $2 * m * n * p$ et pour le stockage on a $m * n$ données.

* question 2.
Pour l'algorithme 8 à 2 boucles sous forme ij en créant une fonction
$$function[C] = matmat2b(A, B)$$
on a :
$p+1$ nombre de multiplications et n additions.
Donc cet algorithme est de niveau 3 c'est à dire de l'ordre $m * n * (p+1)$ et pour le stockage on a $m * n$ données.

Pour l'algorithme 9 à 1 boucle sous forme i en créant une fonction
$$function[C] = matmat1b(A, B)$$
on a :
p nombre de multiplications et $p-1$ additions.
Donc cet algorithme est de niveau 3 c'est à dire de l'ordre $(2p-1) * m * n$ et pour le stockage on a $m * n$ données.

* question 3.
Avec les fonctions $tic()$ et $toc()$ on a mesuré les temps de ces algorithmes pour différentes valeurs.
pour $m = 10; p = 5; n = 15$ on a:
Avec matmat3b : $temps3b = 0.016518$
Avec matmat2b : $temps2b = 0.005871$
Avec matmat1b : $temps1b = 0.005307$

pour $m = 1000; p = 100; n = 1500$ on a:
Avec matmat3b : $temps3b = 0.011457$
Avec matmat2b : $temps2b = 0.018331$
Avec matmat1b : $temps1b = 0.018811$

pour $m = 20; p = 20; n = 20$ on a:
Avec matmat3b : $temps3b = 0.031104$
Avec matmat2b : $temps2b = 0.0135390$
Avec matmat1b : $temps1b = 0.011286$

pour $m = 10000; p = 10000; n = 10000$ on a:
Avec matmat3b : $temps3b = 0.01787$
Avec matmat2b : $temps2b = 0.016683$
Avec matmat1b : $temps1b = 0.018718$

* question 4.
Analysons les résultats.
Le produit matriciel avec 3 ,2 et 1 boucle(s), ils sont tous de niveau 3 et ils font des temps de calculs recevable mais commettent des erreurs si la taille des matrices deviennent de plus

en plus grande donc nos algorithme sont acceptables amis pour de petites tailles car si la taille est grande on va commettre des erreurs numériques très grandes qui peuvent amener à des cancellations c'est le cas avec $n = 10000$

5 Exercice 2. $TP3-Systemetriangulaire$

* question 1.
  On a écrit les algorithmes 10 et 11 de résolution par remontée et descente, en créant deux fichiers *usolve.sci* et *lsolve.sci* avec les en-têtes :

$$function[x] = usolve(U, b)$$

et

$$function[x] = lsolve(L, b)$$

.
Pour l'algorithme de remontée on a:

pour la complexité on trouve :
pour chaque i fixé on fait $n - i$ opérations en multiplication,$n - i - 1$ opérations en addition,1 opération en soustraction et 1 opération en division. et comme i varie de 1 à $n - 1$
donc on a le résultat suivant:

$$\sum_{i=1}^{n-1}(2n - 2i + 1) = n^2 - 1$$

Et pour le stockage on a: $\sum_{i=1}^{n} i = n * (n + 1)/2$

Pour l'algorithme de la descente on a:
pour la complexité on trouve :pour chaque i fixé on fait $i - 1$ opérations en multiplication,$i-2$ opérations en addition,1 opération en soustraction et 1 opération en division. et comme i varie de 2 à n
donc on a le résultat suivant:

$$\sum_{i=2}^{n}(2i - 1) = (n^2 - n)/2$$

et on ajoute 1 opération ce qui nous donne $(n^2 - n)/2 + 1$

Et pour le stockage on a: $\sum_{i=1}^{n} i = n * (n+1)/2$

* question 2.
  Testons et validons nos algorithmes.
  Comparons notre *usolve* avec la fonction *scilab*.
  Pour $n = 100$ en lançant 10 fois le programme pour mesurer le temps d'execution de *usolve* avec celle de *scilab* et on a: pour *usolve* on a: $tempsmoyenusolve = 0.001466$
  pour *scilab* on a: $tempsmoyenscilab = 0.00218$

  Pour $n = 1000$ en lançant 10 fois le programme pour mesurer le temps d'execution de *usolve* avec celle de *scilab* et on a: pour *usolve* on a: $tempsmoyenusolve = 0.010074$
  pour *scilab* on a: $tempsmoyenscilab = 0.5365$

  Pour $n = 10000$ en lançant 10 fois le programme pour mesurer le temps d'execution de *usolve* avec celui de *scilab* et on a: pour *usolve* on a: $tempsmoyenusolve = 1.0258$
  pour *scilab* on a: $tempsmoyenscilab = 701.01$

  Ensuite on calcule l'erreur en avant et après de notre algorithme.
  Pour $n = 100$ on a $erreuravant = 1$ et $erreurapres = 0.634$
  Pour $n = 1000$ on a $erreuravant = 1$ et $erreurapres = 0.603$
  Pour $n = 10000$ on a $erreuravant = nan$ et $erreurapres = 0.608$

  Comparons notre *lsolve* avec la fonction *scilab*.
  Pour $n = 100$ en lançant 10 fois le programme pour mesurer le temps d'execution de *lsolve* avec celui de *scilab* et on a: pour *usolve* on a: $tempsmoyenlsolve = 0.00170$
  pour *scilab* on a: $tempsmoyenscilab = 0.00187$

  Pour $n = 1000$ en lançant 10 fois le programme pour mesurer le temps d'execution de *usolve* avec celui de *scilab* et on a: pour *usolve* on a: $tempsmoyenlsolve = 0.0114$
  pour *scilab* on a: $tempsmoyenscilab = 0.536$

  Pour $n = 10000$ en lançant 10 fois le programme pour mesurer le temps d'execution de *usolve* avec celui de *scilab* et on a: pour *usolve* on a: $tempsmoyenlsolve = 1.04$
  pour *scilab* on a: $tempsmoyenscilab = 819.01$

  Ensuite on calcule l'erreur en avant et après de notre algorithme.
  Pour $n = 100$ on a $erreuravant = 1$ et $erreurapres = 0.598$

Pour $n = 1000$ on a $erreuravant = 1$ et $erreurapres = 0.604$
Pour $n = 10000$ on a $erreuravant = nan$ et $erreurapres = 0.610$

Conclusion pour exercice $2tp3$.
On a vu que nos algorithmes de remontée et de descente sont de niveau 2 et ont un temps de calcul satisfaisant mais commettent des erreurs très grandes par rapport à celui de *scilab* qui de l'ordre de 10^{-16}.Et tout cela aura un impacte si on veut travailler avec des matrices de très grandes tailles sur l'approximation de la solution du problème .

6 Exercice 3. $TP3 - Gauss$

* question 1.
On a écrit l'algorithme de résolution par élimination de Gauss sans pivotage en créant un fichier *gausskij3b.sce* et la fonction

$$function[x] = gausskij3b(A, b)$$

.
Pour l'algorithme de Gauss sans pivot on a:

pour la complexité on trouve :
pour la boucle interne j ,on a pour chaque j fixé on 1 opération en soustraction et 1 opération en multiplication donc 2 opérations pour chaque j, donc cette on $2(n-k)$ opérations .
Pour la boucle interne i pour chaque i on a 1 division,1 opération en soustraction et 1 opération en multiplication. Donc pour cette boucle on fait au total $3(n-k)$ opérations et $2(n-k)^2$ pour la boucle j.
Et comme k varie de 1 à $n-1$
donc on a le résultat suivant:

$$\sum_{k=1}^{n-1}(3(n-k) + 2(n-k)^2) = 2n^3/3 + 5n^2/6 - 3n/2$$

* question 2.
Testons et validons notre algorithme sur de petites matrices.
On compile notre programme 10 fois et on a: pour $n = 20$ on a: $tempsmoyen = 0.00961$ et $erreuravant = 2.19 * 10^{-15}$ et $erreurarriere = 4.75 * 10^{-17}$.
pour $n = 5$ on a: $tempsmoyen = 0.000578$ et $erreuravant = 5.78 * 10^{-16}$ et $erreurarriere = 4.02 * 10^{-17}$.

Et l'algorithme de *scilab* fait en moyenne $tempsmoyen = 4.9 * 10^{-5}$
Conclusion exercice $3TP - Gauss$.
On voit que notre est de niveau 3 et fait les calculs en moyenne pour les petites matrices à l'ordre de $9.61 * 10^{-3}$ en temps et commet des erreurs en avant et arrière de l'ordre $5.78 * 10^{-16}$ et $4.02 * 10^{-17}$.
Donc notre algorithme est recevable pour des matrices de petites tailles mais pour mieux comprendre réellement on doit faire appel au conditionnement.

7 Exercice 4. $TP3 - LU$

* question 1.
On a écrit l'algorithme de factorisation LU en créant le fichier $mylu3b.sce$ avec la fonction

$$function[L, U] = mylu(A)$$

Pour l'algorithme de factorisation avec trois boucles on a:

pour la complexité on trouve :
pour la boucle interne j ,on a pour chaque j fixé on 1 opération en soustraction et 1 opération en multiplication donc 2 opérations pour chaque j, donc cette on $2(n-k)$ opérations .
Pour la boucle interne i pour chaque i on a 1 division. Donc pour cette boucle on fait au total $(n-k)$ opérations et $2(n-k)^2$ pour la boucle j.
Et comme k varie de 1 à $n-1$
donc on a le résultat suivant:

$$\sum_{k=1}^{n-1}((n-k) + 2(n-k)^2) = 40n^3/6 - n^2/2 + 11n/6$$

* question 2.
Testons et validons notre algorithme sur de petites matrices.

On compile notre programme 10 fois et on a: pour $n = 5$ on a: $tempsmoyen = 0.0004061$ et l'erreur moyenne commise qui $norm(A - LU)$ donc $erreurmoy = 7.10852 * 10^{-16}$

* question 3.
On a amélioré l'algorithme de factorisation LU de sorte à n'obtenir qu'une boucle en créant aussi un fichier *mylu1b.sce* et la fonction

$$function[L, U] = mylu1b(A)$$

pour la complexité on trouve :
pour chaque k fixé on a $n-k$ opérations en soustraction et $(n-k)^2$ opérations en multiplication et $n-k$ opérations en division.
Et comme k varie de 1 jusqu'en $n-1$.
Donc on obtient au total le résultat suivant pour cet algorithme:

$$\sum_{k=1}^{n-1}(2(n-k) + (n-k)^2) = n^3/3 - n^2/2 - 5n/6$$

.
Et on fait des tests pour cet algorithme et on a:
pour $n = 5$ on a: $tempsmoyen = 18.43*10^{-5}$ et l'erreur moyenne commise qui $norm(A - LU)$ donc $erreurmoy = 1.919 * 10^{-17}$

8 Conclusion

À travers ces travaux pratiques et dirigés,on a implanté différents algorithmes pour résoudre le problème du produit matriciel et du système linéaire.
Mais on confronte à des problèmes comme: le temps de calcul,l'occupation en mémoire et les erreurs commises.
Donc à travers cela on peut en tirer qu'un bon algorithme est celui réduit son temps de calcul, commet mois d'erreur et occupe moins de mémoire et pour cela en évitant les boucles internes .