$TD/TP4/5$ Calcul Numérique

December 27, 2021

Nom et Prénom:Diop Elhadji Fallou
Numéro Étudiant: 22101564

1 https://$github.com/elzodp/Rapporttp_cal_num$

Part I
$TP4$: Exploitation des structures et calcul creux

2 Introduction

Dans les chapitres précédents,on a vu comment résoudre le système $Ax = b$ pour une matrice quelconque avec la factorisation LU de Gauss .Mais cette méthode nécessite du temps et de la mémoire .C'est dans ce sillage qu'on va essayer de voir d'autres méthodes pour des matrices à structure particulière avec des formats de stockages différentes.

3 Factorisation LDL^T pour une matrice symétrique.

L'existence et l'unicité de la factorisation LU donne l'existence et l'unicité de la factorisation LDL^T d'une matrice A symétrique avec L une matrice triangulaire inférieure avec des 1 sur la diagonale ,D une matrice diagonale et L^T la transposée de L donc triangulaire supérieure.

3.1 Complexité de l'algorithme LDL^T , Test et Validation

Pour la boucle interne i on a 1 opération en multiplication donc cette boucle on fait $j - 1$ opérations.

pour la boucle j on a 1 opération en division,$(j-1)^2$ opérations en multiplication et 2 opérations en soustraction.
Donc pour on a $j-1+3+(j-1)^2$ élémentaires.
Donc le nombre d'opération élémentaire pour cet algorithme est:

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

Stockage mémoire: La matrice A étant symétrique donc on peut stocker qu'une partie donc $n^2/2$

Testons et validons notre algorithme:
En lançant le programme 10 fois de suite et on fait la moyenne :
on a pour $n=3$:

$$tempsmoyenldl^t = 2.39*10^{-04}$$

pour l'erreur on utilise la fonction $lu()$ de *scilab* et la $norm(ldlt-lu,2)$ et on a:

$$erreurmoyenldl^t = 6.21448*10^{-16}$$

on a pour $n=10$:

$$tempsmoyenldl^t = 3.54^{-04}$$

pour l'erreur on utilise la fonction $lu()$ de *scilab* et la $norm(ldlt-lu,2)$ et on a:

$$erreurmoyenldl^t = 1.44298*10^{-15}$$

on a pour $n=100$:

$$tempsmoyenldl^t = 3.92^{-04}$$

pour l'erreur on utilise la fonction $lu()$ de *scilab* et la $norm(ldlt-lu,2)$ et on a:

$$erreurmoyenldl^t = 1.71082*10^{-15}$$

Conclusion pour exercice 1.
On a vu que pour une matrice symétrique A l'algorithme de la factorisation de A en LDL^T a pour complexité de $n^3/3+O(n^2)$.

Et comme la matrice A donc on stocke qu'une partie donc on gagne de l'espace par rapport à la factorisation de Gauss qu'on a vu et le temps moyen de calcul est de l'ordre 10^{-04} qui est 2 fois plus vite que la factorisation de Gauss.
Son erreur par rapport à la fonction $lu()$ de *scilab* est de l'ordre de 10^{-15} donc fait moins d'erreur que la factorisation de Gauss.
Donc on peut conclure que si on travaille avec des matrices symétriques utiliser la factorisation LDL^T plus avantageux que celle de Gauss en temps et en mémoire.

4 Factorisation de $Choleski$ en LL^T.

Dans la section précédente on a vu pour une matrice symétrique on peut faire sa factorisation LDL^T. Maintenant on va aller plus en ajoutant une condition supplémentaire.
Si on a une matrice symétrique définie positive alors on a l'existence et l'unicité de la factorisation de $Choleski$ en LL^T avec L une matrice triangulaire.

4.1 Complexité de l'algorithme LDL^T , Test et Validation

Pour la boucle interne i on a 1 opération en multiplication donc cette boucle on fait $n+1$ opérations.
pour la boucle j on a $2(n-j+1)(j-1)$ opérations en multiplication , $(n-j+1)$ opérations en addition et 2 indices.
Donc pour cette boucle on a $2(n-j+1)(j-1)+(n-j+1)+2$ élémentaires.
Donc le nombre d'opération élémentaire pour cet algorithme est:

$$n+1+\sum_{j=2}^{n}(2(n-j+1)(j-1)+(n-j+1)+2) = n+1+\sum_{j=2}^{n}((n-j+1)(2(n-j)+2)+3) = n^3/3+n^2+8n/3-2$$

Testons et validons notre algorithme:
En lançant le programme 10 fois de suite et on fait la moyenne :
on a pour $n = 3$:

$$tempsmoyenll^t = 1.786 * 10^{-04}$$

pour l'erreur on utilise la fonction $chol()$ de *scilab* et la $norm(llt - chol(), 2)$ et on a:

$$erreurmoyenll^t = 1.0143$$

on a pour $n = 10$:

$$tempsmoyenll^t = 7.271^{-04}$$

pour l'erreur on utilise la fonction $chol()$ de $scilab$ et la $norm(llt-chol, 2)$ et on a:

$$erreurmoyenll^t = 4.368$$

on a pour $n = 100$:

$$tempsmoyenll^t = 0.06588$$

pour l'erreur on utilise la fonction $chol()$ de $scilab$ et la $norm(llt-chol, 2)$ et on a:

$$erreurmoyenll^t = 49.50652$$

Conclusion pour exercice $1.tp4$.
On a vu que pour une matrice symétrique définie positive A l'algorithme de la factorisation de A en LL^T a pour complexité de $n^3/3 + O(n^2)$.
Et comme la matrice A est symétrique définie positive donc on stocke qu'une partie donc on gagne de l'espace par rapport à la factorisation de Gauss qu'on a vu et le temps moyen de calcul est de l'ordre 10^{-04} qui est 2 fois plus vite que la factorisation de Gauss.
Son erreur par rapport à la fonction $lu()$ de $scilab$ est de l'ordre de 10^{-15} on donc fait moins d'erreur que la factorisation de Gauss.
Et aussi par rapport à la factorisation LDL^T, la factorisation LL^T prime car et terme algorithmique il est meilleur. Donc on peut conclure que si on travaille avec des matrices symétriques définies positives utiliser la factorisation LL^T plus avantageux que celle de Gauss en temps et en mémoire.

5 exercice $5.tp4$: Produit Matrice vecteur creux.

Dans un premier,on va parler le format CSR.

5.1 Format COO et CSR

Principe:
Le format csr est spécialisé dans les opérations d'algèbres linéaires et pallie les défauts du format COO qui permet de stocker une matrice sous forme de trois

tableaux en row,col et val et les trois de tailles de nnz et contenant respectivement l'indice ligne,colonne et le coefficients non nuls de la matrice.
Pour $i = 0$ jusqu'à $(nnz - 1)$.
On a : $A(row[i], col[i])$=$val[i]$
Le format COO peut autoriser les doublons c'est à dire les coefficients ayant les meme valeurs.
Pour le csr le tableau row est compressé.
Une matrice au format csr est composé des deux tableaux col et val comme pour le coo et ordonnés par "lignes" et le tableau row est définit ainsi .
Sa taille est $n + 1$ (n lignes)
$Row[i]$ est maintenant l'indice du premier élément non nul de la ligne i dans les tableaux col et val. Le tableau row est compressé par rapport au format coo puisque sa taille est maintenant de $n + 1$ bien inférieure à nnz .
Sur une petite matrice,le gain de mémoire est très faible ,mais sur une grande matrice à plusieurs millions d'entée ,cette stratégie devient payante.D'autre part ,l'absence de doublons de coefficient le fait que les tableaux sont triés permettant d'améliorer significativement les opérations d'algèbres.
soit la matrice A de 6 lignes et 8 colonnes on donne le format csr est:

$$A = \begin{pmatrix} 15 & 0 & 0 & 22 & 0 & -15 & 0 & 0 \\ 0 & 11 & 3 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 & 25 & 7 \\ 0 & 0 & 28 & 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$

on a :

$JA =$

1	4	6	2	3	7	4	1	7	8	3	8

$AA =$

15	22	-15	11	3	2	-6	91	25	7	28	-2

$IA =$

1	4	7	8	8	11	13

Le format csr est rigide, il est très couteux d'ajouter des élément dans la matrice .Ainsi et afin de ne pas perdre en efficacité, il est nécessaire de connaitre à l'avance l'emplacement des coefficients non nuls de la matrice avant de la construire . En revanche ,une fois construire cette forme de stockage est très efficace.

5.2 Produit Matrice vecteur creux

Le produit matrice vecteur est couteux en calcul numérique.Donc pour pallier à cela on va essayer une méthode pour faire le produit matrice vecteur pour des

matrices particulières comme les matrices creuses en utilisant le format csr donc si on prend matrice A précédente avec les vecteurs suivants :

$$x1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

et on a le résultat suivant:

$$y1 = \begin{pmatrix} 22 \\ 16 \\ -6 \\ 0 \\ 123 \\ 26 \end{pmatrix}$$

et son temps de calcul moyen est :

$$temps1 = 2.35 * 10^{-04}$$

Et pour le vecteur $x2$:

$$x2 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

et on a obtenu $y2$:

$$x1 = \begin{pmatrix} 15 \\ 5 \\ 0 \\ 0 \\ 116 \\ 28 \end{pmatrix}$$

Son temps de calcul moyen est :

$$temps2 = 2.24 * 10^{-04}$$

conclusion: Pour une matrice dense de n lignes et de m colonnes donc pour son

stockage on doit utiliser si on est double précision $8 * n * m$ en mémoire et cela peut nous couter très chère si on utilise des matrices de tailles très grandes donc pour faire le produit matrice vecteur cela aussi va nous couter très chère. Pour pallier tout cela avec des matrices particulières comme les matrices creuses on pourra stocker que les éléments non nuls et avec le format csr on pourra avoir un gain en mémoire surtout pour les matrices de grandes tailles.

Part II

$TP5$: Résolution de l'équation de la chaleur

6 Introduction

Dans les cas précédents, on a vu des formats de stockage et le produit matrice creuse vecteur.Maintenant on essaie de résoudre l'équation de la chaleur pour cela on a besoin d'autres formes de stockage, les appels des bibliothèques $BLAS$ et $LAPACK$ et des méthodes pour la résolution de l'équation $Ax = b$.
Les méthodes on a : *Jacobi* , *Gauss-Seidel* et *Richardson*.

7 Travail préliminaire, Problème de la chaleur ,Utilisation de BLAS/LAPACK, stockage bande, stockage row-major et col-major,dgbsv,dgbmv

Considérons l'équation de la chaleur dans un milieu isotrope,immobile linéaire homogène avec les termes sources.
Pour cette équation on utilise la méthode de différence finie centrée d'ordre 2.
On utilise la discrétisation en 1D du domaine pour un pas constant h et de $n+2$ noeuds des x_i pour $i = 0, 1, 2, ..., n+1$ et en chaque noeud on a une équation aux dérivées partielles de la température aux sources en chaque noeud donc on au aura en tout n équations et n inconnues.On peut caractériser le système comme cela $Au = f$ avec A une matrices d'ordre n et u et f sont des vecteurs d'ordre n.
Si on considère qu'on n'a pas de source de chaleur et la solution qu'on peut obtenir analytiquement est :

$$T(x) = T_0 + x(T_1 - T_0)$$

.
Et on obtient la matrice suivant pour l'équation de la chaleur :

$$A_n = \begin{pmatrix} 2 & -1 & 0 & 0 & ... & ... & 0 & 0 \\ -1 & 2 & -1 & ... & ... & ... & 0 & 0 \\ 0 & -1 & 2 & -1 & ... & ... & 0 & 0 \\ 0 & ... & -1 & 2 & ... & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... & -1 & 0 \\ 0 & ... & ... & ... & ... & -1 & 2 & -1 \\ 0 & 0 & 0 & ... & 0 & 0 & -1 & 2 \end{pmatrix}$$

Cette matrice de n lignes et n colonnes a n valeurs propres données par,pour $k = 0, 1, 2, 3..., n$:

$$\lambda_k = 4\sin^2(\frac{k\pi}{2(n+1)})$$

En notant $L := M^{-1}(M - A) = I - M^{-1}A$ la matrice itérative des méthodes de Jacobi,Gauss-Seidel et de Relaxation et $\rho(L)$ son rayon spectral.
Pour atteindre une erreur relative de ϵ , le nombre d'itération estimé est donné par la formule suivant:

$$n_{iter} = \frac{\ln(\epsilon)}{\ln(\rho(L))}$$

,donc la connaissance du rayon spectral de la matrice d'itération peut nous donner le nombre d'itération nécessaire pour la convergence.
La matrice de l'équation de la chaleur ci-dessus est une matrice tridiagonale donc une matrice bande donc on doit faire appel à d'autres stockages pour le gain de temps et de la mémoire.
En faisant appel aux bibliothèques BLAS et LAPACK nous utilisons le stockage Genearal Band car on a une bande supérieure et une bande inférieure, dont leur somme est inférieure à la taille de la matrice.
Fonctionnement de ce mode de stockage: Soit une matrice A de m lignes et n colonnes.Donc on suppose :
kl le nombre de sous-diagonal.
ku le nombre de sur-diagonal.
Avec le stockage General Band ,cette matrice peut etre stocker de manière compacte dans un tableau à deux dimensions de $kl + ku + 1$ lignes et de n colonnes.
Ce mode de stockage fonctionne que si kl et ku sont tous inférieurs au $\min(m, n)$.
Les colonnes de la matrice A sont stockés dans les colonnes correspondantes du tableau et les diagonales de A stockées dans les lignes du tableau.
Pour ce faire LAPACK donne la terminaison B à la matrice transformée et on aboutit à la formule suivant de passage de la matrice A à la matrice AB :

$$AB(ku + i + 1 - j, j) = A(i, j)$$

.
Dans le cas de la matrice A issue de l'équation de la chaleur son stockage en

General Band est la suivante:

$$AB = \begin{pmatrix} * & -1 & -1 & ... & ... & ... & -1 & -1 \\ 2 & 2 & ... & ... & ... & ... & 2 & 2 \\ -1 & -1 & ... & ... & ... & ... & -1 & * \end{pmatrix}$$

Les éléments marqués $*$ doivent etre affectés selon les utilisations de la matrice mais notre cas on les affecte la valeur 0 .
Après avoir stockée la matrice A de l'équation de la chaleur en General Band on peut ensuite faire appel à d'autres stockages comme le fait de stocker les éléments de la matrice AB en ligne ou bien en colonne.
Stockage en Row-major à une dimension:
La i-ème ligne du tableau AB correspond à la i-ème élément du tableau $ABRM$.
Donc pour la matrice A de l'équation de la chaleur on a son stockage en row-major suivant:

$$AB1DROWM = \begin{bmatrix} 0 & -1 & -1 ... & -1 & 2..2...2 & -1 ... & -1 ... & -1 & 0 \end{bmatrix}$$

Donc on se retrouve avec un tableau de 3 éléments mais chaque élément pointe sur n éléments.
Et pour le stockage en Col-major à une dimension on a :
La j-ème colonne AB correspond à la j-ème élément du tableau $ABCM$.
Pour notre matrice A de l'équation de la chaleur on a son stockage en col-major suivant:

$$AB1DCOLM = \begin{bmatrix} 0 & 2 & -1 & ... & -1 & 2 & -1 & ... & -1 & 2 & 0 \end{bmatrix}$$

Donc on se retrouve avec un tableau de n éléments mais chaque élément pointe sur 3 éléments.
Donc les deux formats ont des avantages et des inconvénients selon le langage utilisé car le langage C utilise le stockage ligne principal des tableau tandis que le langage $Fortran$ utilise le stockage colonne des tableaux.
Pour le langage C,l'allocation de la mémoire d'un tableau à deux dimensions dépend du nombre ligne,de colonne et du type de données utilisé(int, float,double,..) car en C les tableaux sont considérer comme des pointeurs donc nécessite une allocation de mémoire et la syntaxe est la suivante : Par exemple en double précision on a pour une matrice A de m lignes et n colonnes :

$$A = (double^*)malloc(sizeof(double) * m * n)$$

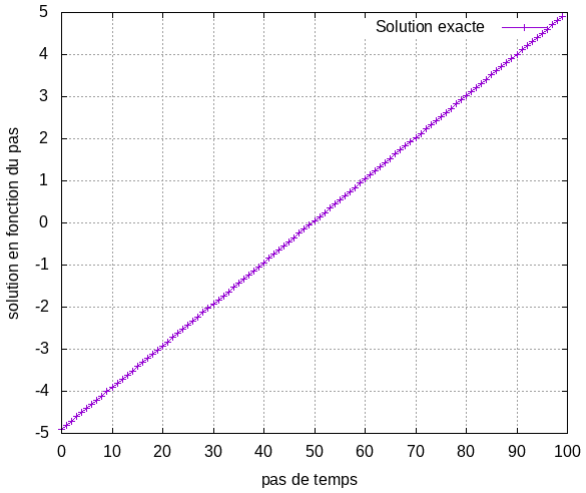
.
La constante LAPACK-ROW-MAJOR ou LAPACK-COL-MAJOR sont définis dans *lapacke.h* et spécifiant si les tableaux sont stocker dans l'ordre des lignes ou des colonnes. Cette constante est utilisée par LAPACKE(l'interface C de LAPACK) Pour déterminer si le bloc de mémoire qu'on transmet(via un pointeur en C) fait référence à la mémoire organisée par ligne principale(toute une ligne précède la ligne suivante) ou par colonne majeure (toute une colonne précède

la colonne suivante).ET ceci se généralise pour plus de tableau en 2D avec row "outer most dimension" et col"inner most dimension".
Ainsi cette constante détermine l'interprétation de la mémoire,mais on doit toujours indiquer la forme aux routines LAPACKE(via un argument généralement appelé $intN$ dans LAPACK et LAPACKE) La dimension principale d'une matrice A est supérieure ou égale à la somme dU double de kl du nombre sous-diagonal, du ku nombre de sur-diagonal et 1.Et on a:

$$ld >= kl + ku + 1$$

Par exemple des données de ligne principale,cela répond " de combien d'éléments dois-je avancer pour passer d'une ligne i à l'autre $i + 1$.
La sous-routine $DGBSV(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO)$ est une routine de pilote LAPACK(Version 3.1) crée par l'université du Tennesee,l'université de California Berkley et NAG ltd en novembre 2006.
Les arguments scalaires sont en $Integer$: INFO, KL, KU, LDAB, LDB, N,NRHS.
Avec : N est un entier et donné en entrée. il correspond à l'ordre de la matrice A et il positif ou nul.
KL est un entier et donné.Et c'est le nombre de sous-diagonales dans la bande de A.et il est positif ou nul.
KU est un entier donné en entrée et correspond au nombre de sur-diagonales dans la bande de A.
NRHS est un entier positif ou nul et donné en entrée et c'est le nombre de colonnes de la matrice B.
LDAB est un entier positif ou nul, donné en entrée et correspond à la dimension principale de tableau AB et elle vérifie :$LDAB >= 2 * KL + KU + 1$
LDB est un entier donné en entrée et correspond à la dimension principale du tableau B et vérifie:$LDB >= \max(1, N)$
INFO est un entier et c'est un argument de sortie et il nous donne l'information de la convergence selon les valeurs de sortie. Si la sortie est nulle donc on peut dire que la sortie est réussie Si la sortie est strictement négativement et égale à l'opposé de i donc le i-éme argument avait une valeur illégale. Et si la sortie est strictement positif et égale à i donc $U(i, i)$ est exactement égale à zéro et la factorisation est terminée ,le facteur U est exactement singulier, et la solution n'a pas été calculée.
Les arguments du tableau:
IPIV est un pointeur qui pointe sur un entier et sa dimension est N.Les indices pivots qui définissent la matrice de permutation P; la ligne i de la matrice a été intervertie avec la ligne IPIV(i).
AB est un tableau entre-sorti en double précision de dimension $LDAB * N$ avec le stockage bande défini plus nous donne les éléments de AB en fonction de A.
Et en sortie ,on a les détails de la factorisation:la matrice U est stockée comme matrice de bande triangulaire supérieure.
B est un tableau d'entré sorti en double précision de dimension $LDB * NRHS$,à l'entrée la matrice est N par NRHS de droite B.
La fonction *dgbsv* calcule la solution de d'un système réel d'équations linéaires

$AX = B$. Avec A une matrice bande d'ordre n avec des sous-diagonales kl et des sur-diagonales ku et X et B sont des matrices d'ordre n par NRHS(le nombre de cotés droits, c'est à dire le nombre de colonnes de la matrice B).
cette fonction utilise la décomposition LU avec pivotement partiel et échanges pour factoriser A comme $P * A = L * U$ où P la matrice de permutation,L est un produit de permutation et des matrices triangulaires inférieures unitaires avec kl sous-diagonales et U est triangulaire supérieur avec kl sous-diagonales + ku sur-diagonales.
Et la forme factorisée obtenue est utilisée pour la résolution du système $AX = B$.
Dans $tp2 - poisson - 1D - direct.c$ pour passer de $COL - MAJOR$ à $ROW - MAJOR$ on met juste la constante m à 1, c'est à dire $m = 1$
La constante kv nous permet de faire un produit matrice-vecteur avec $kv = 0$ ou bien de résoudre le système avec $kv = 1$ et on obtient avec les résultats de la résolution du système.En faisant à dgbsv on a vu la constante $INFO = 0$ donc la sortie est réussie c'est on a arrivé à résoudre le système $AX = B$ et l'erreur est de l'ordre $5.889846 * 10^{-15}$ ce qui veut dire les données entrées ont été perturbées et cette erreur correspond à celle de LU de Gauss.



Dans cette courbe on va bien que la solution est linéaire ce qui correspond à notre problème.

Maintenant parlons en du dgbmv. $DGBMV$ calcule le produit matrice-vecteur pour une matrice de bande générale réelle ou sa transposition.La matrice de bande générale est stockée en mode de stockage$GENARALBAND$ de $BLAS$.
Il utilise les scalaires α et β, les vecteurs x et y, et la matrice de bande générale A ou sa transposée, c'est à dire elle résout l'équation $y = \alpha Ax + \beta y$ ou $y = \alpha A^T x + \beta y$ ou bien $y = \alpha A^H x + \beta y$.
Sa syntaxe avec $cblas - dgbmv$ est la suivante:
$cblas - dgbmv(cblas - layout$, $cblas - transa$, m, n, ml, mu, alpha, a, lda, x, $incx$, $beta$, y, $incy$);
Si la matrice d'entrée est stockée dans l'ordre principal des lignes ou des colonnes

alors $cblas{-}layout = CblasRowMajor$ ou bien $cblas{-}layout = CblasColMajor$.
$cblas - transa$ indique la forme de la matrice A à utiliser dans le calcul.
Si $cblas - transa = CblasNoTrans$ alors A est utilisé dans les calculs ou est égale $CblasTrans$ alors A^T est utilisé dans les calculs ou bien est égale à $CblasConjTrans$ alors la transposée de la matrice adjoint de A est utilisée.
La matrice est de taille $m * n$.
Le ml correspond à kl et mu à ku de *dgbsv* Les α et β sont des constantes.
Le lda correspond à ld de *dgbsv*.
Les x et y sont des vecteurs donnés.
Les $incx$ et $incy$ correspondent aux strides pour x et y. Dans notre cas on souhaite résoudre l'équation $Ax = b$ avec dgbmv pour cela on utilise dgbmv col-major avec A la matrice de Poisson,X le vecteur donné en entrée et y correspond à RHS et $\alpha = 1$,$\beta = -1$ et $incx = incy = 1$.
On écrase le résultat dans RHS pour pour pouvoir calculer l'erreur résiduelle relative et on obtient une erreur de l'ordre $4.394808 * 10^{-16}$ qui est une erreur tolérante donc notre méthode peut etre validée. On peut noter que l'utilisation de l'ordre des lignes principales nécessite plus de temps et mémoire celle des colonnes.

8 Methode iterative:Jacobi et Gauss-Seidel

8.1 Jacobi

On cherche à estimer le nombre d'itération pour une tolérance donnée ϵ,pour cette la matrice $M = diag(A)$ ce qui veut dire que $M = \frac{1}{2}I$ avec I la matrice d'identité.
Alors la matrice d'itération $J = I - \frac{1}{2}A$ et les valeurs propres $\phi_k(J)$ de J vérifient donc $\phi_k(J) = 1 - \frac{\lambda_k}{2}$ et $\sin^2(\frac{k\pi}{N+1}) = \frac{1-\cos(\frac{k\pi}{N+1}}{2}$.Donc

$$\phi_k = \cos(\frac{k\pi}{N+1})$$

et le rayon spectral est :

$$\rho(J) = \cos(\frac{\pi}{N+1})$$

et le nombre d'itération nécessaire est :

$$n_{iter}(J) = \frac{\ln(\epsilon)}{\ln(\cos(\frac{\pi}{N+1}))}$$

8.2 Gauss-Seidel

Comme la matrice A de Poisson 1D est tridiagonale alors

$$\rho(G) = \rho^2(J) = \cos^2(\frac{\pi}{N+1})$$

et le nombre d'itération est :

$$n_{iter}(G) = \frac{n_{iter}(J)}{2}$$

.
Pour $n = 3$ et la tolérance $\epsilon = 10^{-1}$ on a : $n_{iter}(J) = \frac{1}{\ln(\cos(\frac{\pi}{4}))} = \frac{1}{\ln(\frac{\sqrt{2}}{2})}$ qui est le double de l'itération de Gauss-Seidel.

n	3	4	5
J	22	29	46
GS	8	15	23

Donc on a vu que si n augmente le nombre d'itération de Jacobi est proche du double de celle de Gauss-Seidel.

9 Conclusion

La résolution du système $Ax = b$ admet plusieurs méthodes mais elles par leurs gains de temps et espace mais celle de Gauss-Seidel prime sur Jacobi car elle plus rapidement la solution mais commet aussi plus d'erreur .

10 Reference

Le cours de Pablo et de Thomas Dufaud
https://www.ibm.com/docs/
https://www.netbi.org/