Projet Krylov : Block GMRES

Quentin Jaubertie

May 2017

Contents

1 Introduction **2**

2 Méthode GMRES **2**
2.1 Principe 2
2.2 Optimisation de l'algorithme 3
2.3 Préconditionnement de A 3

3 Méthode GMRES par blocs **4**
3.1 Principe 4
3.2 Comparaison avec la méthode classique (Flop et mémoire) 5
3.2.1 Mémoire 5
3.2.2 Flops 5

4 Choix de programmation : Block Gmres **6**

5 Tests **7**
5.1 Comparaison entre MyGmres et Block Gmres 7
5.2 Préconditionnement avec la matrice bfw 8
5.3 Résidu 9

1 Introduction

Dans ce projet, nous allons nous intéresser à une méthode de résolution itérative du système linéaire :

$$Ax = b$$

avec A de dimension n. Cette méthode se nomme **GMRES** (i.e. Generalized minimal RESidual). Comme son nom l'indique, elle est basée sur une minimisation du résidu $\|b - Ax_k\|$.
Tout d'abord, nous allons voir le principe de la méthode GMRES classique. Ensuite, nous inclurons un traitement supplémentaire : le préconditionnement de la matrice A. Pour finir, nous généraliserons cette méthode avec des sytèmes sous la forme :

$$A[x_1, \cdots, x_s] = [b_1, \cdots, b_s]$$

Nous utiliserons la méthode GMRES de matlab afin de comparer l'efficacité de nos méthodes.

2 Méthode GMRES

2.1 Principe

Cet algorithme cherche à itérer le vecteur $x_m = x_0 + K(A, b - Ax_0, m) = x_0 + Span(b - Ax_0, A(b - Ax_0), \cdots, A^{m-1}(b - Ax_0))$ tout en minimisant le résidu $\|b - Ax_0\|$.
L'algorithme est basé sur la relation d'Arnoldi :

$$\boxed{\mathrm{AV}_m = V_{m+1}\overline{H}_m}$$

avec :

- $V_m = [v_1, \cdots, v_m]$ une base orthogonale de l'espace de Krylov $K(A, r_0, m)$
- $\overline{H}_m$ Hessenberg supérieure telle que $\overline{H}_m(1 : end - 1, :) = V_m^T A V_m$

Grâce à cette dernière, on aura un itéré sous la forme : $\boxed{\mathrm{x}_m = x_0 + \sum_{i=1}^{m} y_i v_i = x_0 + V_m y_m}$ avec :

- V_m la base cité ci-dessus
- $y_m = argmin\|b - Ax_m\|_2 = argmin\|\beta e_1 - \overline{H}_m y_m\|_2$
- $\beta = \|r_0\|$

L'algorithme va donc s'organiser de la manière suivante :

```
while(residu/||b|| > epsilon)
     Construction de H et Vm ( Gram-Schmidt )
     Calcul de ym = Hm\(beta*e1)
     Calcul de xm = x0 + Vm*ym
     Calcul du résidu ||b-Axm||
end while
```

2.2 Optimisation de l'algorithme

Le principal défaut de cette version est que l'on doit calculer l'itéré et son résidu à chaque itération. Cependant, il existe une astuce qui permet d'alléger le coût calculatoire de l'algorithme. En effet, en calculant Q_m et $\overline{R}_m$ les matrices de la factorisation QR de la matrice $\overline{H}_m$, on montre que:

$$y_m = [\overline{R}_m(1 : end-1, :)]^{-1}\overline{g}_m(1 : end-1)$$

avec

$$\overline{g}_m = \beta Q_m^T e_1$$

De plus :

$$\boxed{\|b - Ax_m\| = |\overline{g}_m(end)|}$$

Ceci implique que nous ne sommes plus obligés de calculer l'itéré explicitement à chaque itération. La nouvelle version de l'algorithme sera :

```
while(residu/||b|| > epsilon)
     Construction de H et Vm ( Schmidt )
     [Qm,Rm] = Qr(H)
     Calcul de gm = beta Qm'*e1
     Calcul du résidu gm(end)
end while
Calcul de ym = Rm(1:end-1,:)\gm(1:end-1)
Calcul de xm = x0 + Vm*ym
```

2.3 Préconditionnement de A

Considérons deux matrices M_1 et M_2 de même taille de A et M = M_1M_2. On va maintenant appliquer l'algorithme sur la matrice $M^{-1}A$ et $M^{-1}b$ de telle sorte que :

- La convergence soit accélérée.
- M soit facile à manipuler.
- " MA $\approx$ I "

Pour ce projet, nous allons considérer 3 préconditionnements :

- La factorisation LU incomplète de A
- La factorisation de Cholesky incomplète de A
- Préconditionnement de Jacobi

Si le préconditionnement est correct, le nombre d'itérations sera fortement réduit. Voici un exemple de préconditionnement sur la matrice mat1 :

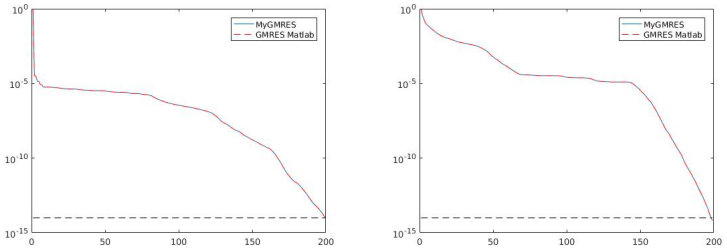

Figure 1: Tracé du résidu GMRES sans préconditionnement et avec préconditionneur de Jacobi

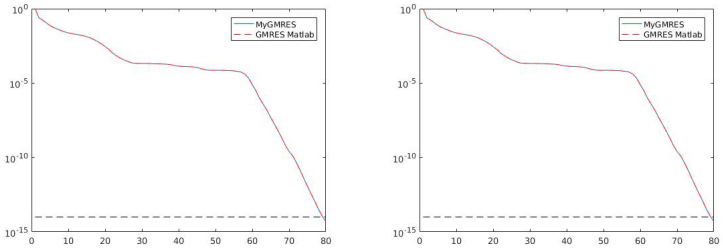

Figure 2: Tracé du résidu GMRES avec préconditionneur de Cholesky et LU

On remarque que, pour les deux derniers exemples, le nombre d'itérations est considérablement réduit. De plus, on constate que les résidus trouvés sont identiques à ceux calculés par Matlab. En pratique, on ne calculera pas M^{-1} mais on utilisera la commande suivante : $\boxed{\text{M}_2\backslash(M_1\backslash mat)}$

3 Méthode GMRES par blocs

3.1 Principe

Le principe de cette méthode est le même que la méthode GMRES mais appliquer à des systèmes matricielles sous la forme :

$$A[x_1, \cdots, x_s] = [b_1, \cdots, b_s]$$

Cette méthode est donc une généralisation du cas précédent. On pourrait simplement se contenter d'appliquer l'algorithme précédent sur chaque système $Ax_i = b_i, \forall i \in$ [1,s], mais le fait de tout calculer directement devrait accélérer la convergence.
Voici les analogies faites avec la partie précédente :

- $V_m = [v_1, \cdots, v_m]$ avec v_i une matrice de taille n*s (au lieu d'un vecteur)
- $\overline{H}_j = (H_{i,k})$ avec H_{ij} une matrice de taille p*p (au lieu du réel H(i,j))
- A l'itération j, $E_1 =$ les s premiers vecteurs de $I_{(j+1)*s}$ (au lieu du premier vecteur e_1)
- β correspond à R_1 avec $R_0 = b - AX_0 = V_1 R_1$ la factorisation QR incomplète de R_0
- Le résidu sera $\|b - AX_m\|_{Fro}$ (On n'utilise plus la $\|.\|_2$)

Pour la version optimisée, on fera les analogies suivantes :

- $\overline{g}_m = Q_m^T E_1 R1$ avec $[Q_m, R_m] = Qr(H)$ (une matrice au lieu d'un vecteur)
- Le résidu vaut $\|\overline{g}_m(end, :)\|_{Fro}$ (au lieu de $|\overline{g}_m(end)|$)
- $y_m = [R_m(1 : end - s, :)]^{-1}\overline{g}_m(1 : end - s, :)$
- $x_m = x_0 + [v_1, \cdots, v_m]y_m$, avec v_i une matrice de taille n*s

3.2 Comparaison avec la méthode classique (Flop et mémoire)

3.2.1 Mémoire

Pour la méthode GMRES classique, on stocke principalement les matrices V et H. A l'itération j, on a :

- Matrice V de taille **Length(b)*(j+1)**=n(j+1)
- Matrice H_1 de taille **(j+1)*j**

Pour la méthode GMRES Block, la matrice W devient aussi importante. Ces matrices sont de taille (en notant p le nombre de second membres) ;

- Matrice V de taille **size(b,1)*((j+1)p)** = n*((j+1)p)
- Matrice H de taille **((j+1)p)*(jp)**
- Matrice W de taille **Size(A,1)*p**

Pour la deuxième méthode, on a donc p^2 matrices H_1 de la première méthode donc le coût de stockage est nettement plus important pour p grand.

3.2.2 Flops

Soit n = size(A,1).
A l'itération j, la méthode GMRES classique coûte :

- Gram-Schmidt : $\approx$ 4j*n
- Produit Matrice-Vecteur : 2nnz(A)-n
- Factorisation QR de H : $2(j+1)j^2$-$\frac{2j^3}{3}$

Pour m itérations, on a :

$$\boxed{\text{Flops} \approx 2m(m+1)n + 2m * nnz(A) - mn + 2\sum_{j=1}^{m}[(j+1)j^2 - \frac{j^3}{3}]}$$

Pour Block GMRES, on a à l'itération j :

- Le produit Wj=A*Vj : Coût $\approx n^2p$
- Factorisation QR de Wj (de taille n*p) : $2np^2$-$\frac{2p^3}{3}$
- Factorisation QR de H (de taille ((j+1)p)*(jp)) : 2p(j+1)$(jp)^2$-$\frac{2(jp)^3}{3}$
- Gram-Schmidt : $\approx$ 4(pj)*n

Pour m itérations, on a :

$$\boxed{\text{Flops} \approx mn^2p + 2mnp^2 - \frac{2mp^3}{3} + 2\sum_{j=1}^{m}[p(j+1)(pj)^2 - \frac{(pj)^3}{3}] + 2pnm(m+1)}$$

Par exemple, pour A=mat1 :

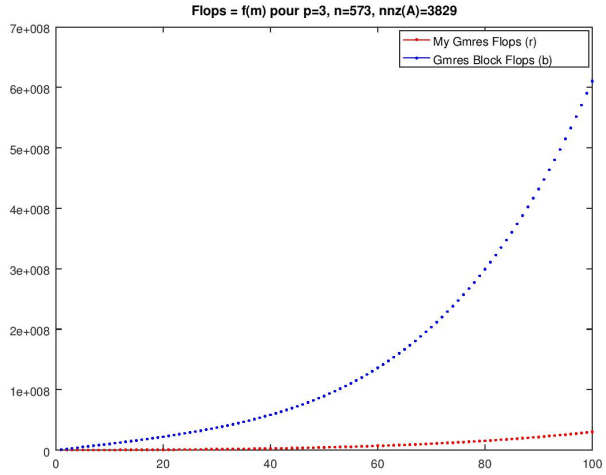

Figure 3: Allure du nombre de Flops en fonction de m

Ce graphique montre bien que la méthode Block Gmres est beaucoup plus couteuse en terme de Flops. Cependant, le nombre d'itérations de GMRES Block est censé être considérablement inferieur à celui de MyGmres.

4 Choix de programmation : Block Gmres

Dans cet algorithme, j'ai stocké les différentes matrices H_{ij} dans une seule matrice bidimensionnelle :

$$\boxed{\text{H((i-1)p+1:ip , (j-1)p+1:jp)} = \text{H}_{ij}}$$

Cette notation est lourde au niveau des indices mais elle me semblait plus pratique que d'ajouter une autre dimension à la matrice de stockage. Il en est de même pour la matrice V (Base orthogonale) qui reste bidimensionnelle :

$$\boxed{\text{V((i-1)p+1:ip , (j-1)p+1:jp)} = \text{v}_i}$$

avec chaque V_i de dimension n*p. Elle permet notamment de garder la même convention que H (Même si ajouter une dimension à la matrice V aurait été plus pratique).

5 Tests

5.1 Comparaison entre MyGmres et Block Gmres

Les second membres sont pris aléatoirement. Pour A = mat0, p $\leq$ 15, sans préconditionnement :

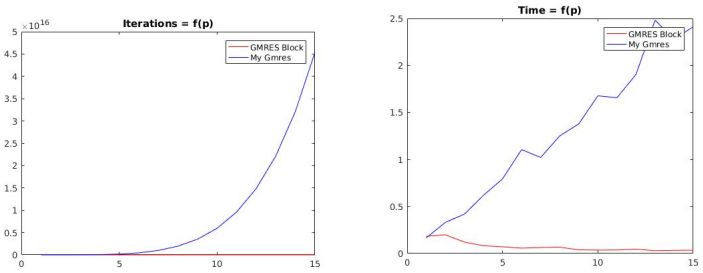

Figure 4: Flops et Temps en fonction de p

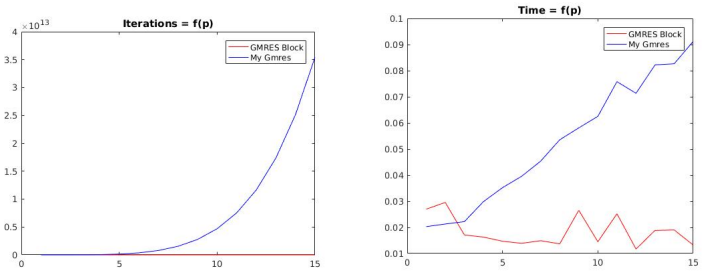

Figure 5: Flops et Temps en fonction de p avec préconditionnement LU

On remarque que le temps pour Block Gmres décroit ce qui et anormal. Je pense que ceci est du au fait que le temps de calcul est très court (≈ 0.03) et du coup oscille. Il est alors préférable de réaliser plusieurs fois cette experience et de prendre la moyenne des temps.

Hormis ce défaut, on constate que la méthode Block GMRES est beaucoup plus efficace en temps et en flops. Plus le nombre de second membre augmente, moins MyGmres est efficace et plus l'écart se creuse entre les deux méthodes.

5.2 Préconditionnement avec la matrice bfw

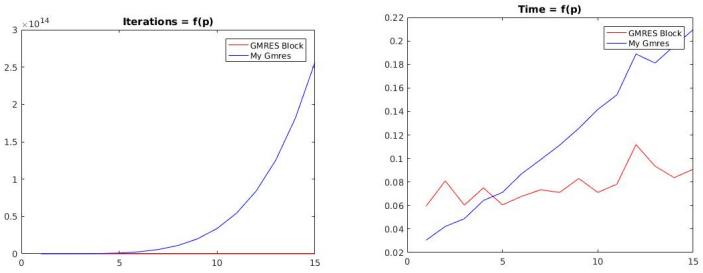

Figure 6: Flops et Temps en fonction de p avec préconditionnement LU pour la matrice bfw

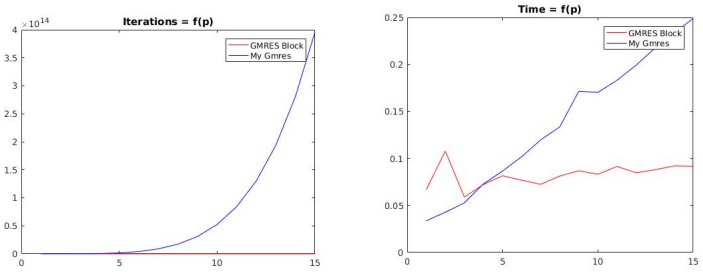

Figure 7: Flops et Temps en fonction de p avec préconditionnement Cholesky sans fill-in pour la matrice bfw

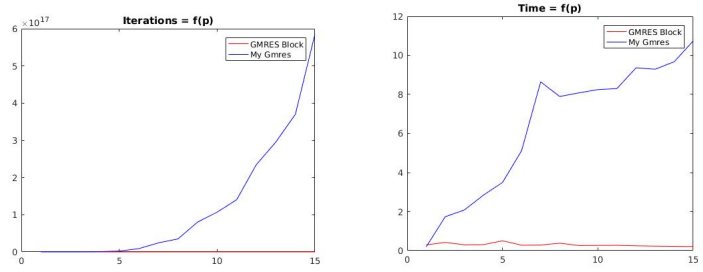

Figure 8: Flops et Temps en fonction de p avec préconditionnement Jacobi pour la matrice bfw

Le but de cette partie est de tester l'efficacité des préconditionneurs. D'après les tests, il semble que le préconditionneur de Jacobi soit le moins efficace. En effet, le nombre de flops explose jusqu'à $6*10^{17}$ pour la méthode MyGmres alors que le temps augmente jusqu'à 11s. En utilisant les préconditionneurs LU et Cholesky, le nombre de flops maximum est divisé par 1000 ce qui est non négligeable. De même, le temps maximum de MyGMres passe de 12s à 0.2s. Il semble tout de même que la factorisation LU incomplète soit superieure à Cholesky que ce soit en temps ou en flops. Pour finir, nous voyons encore la superiorité de la méthode Block Gmres. Quelque soit le préconditionneur, son temps d'exécution ne dépasse pas les 0.2s et son nombre de flops reste négligeable comparé à celui de MyGmres.

5.3 Résidu

Ici, p = 20, mat = mat0 :

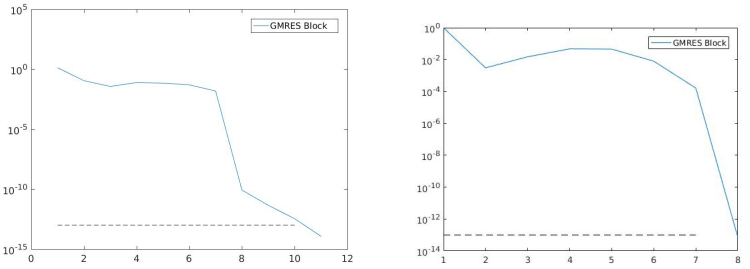

Figure 9: Norme du résidu sans préconditionnement et avec Jacobi

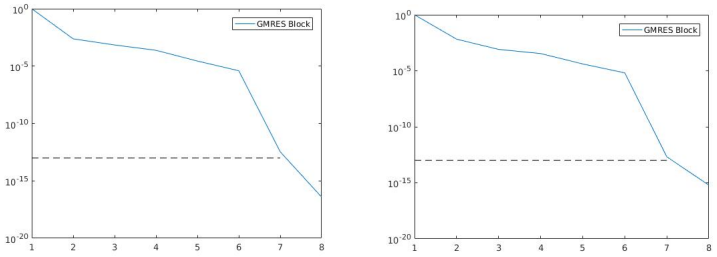

Figure 10: Norme du résidu avec Cholesky et avec LU

Cette partie permet d'étudier l'évolution de $\|b - Ax_m\|$ pour 20 second membres et avec différents préconditionnements. Tout d'abord, nous constatons que le préconditionnement permet d'enlever 4 itérations. Sans conditionnement, on remarque que la norme du résidu "stagne" de la deuxième à la septième itération. Les itérés calculés ne font donc pas progresser l'algorithme. En utilisant la factorisation LU ou Cholesky, on constate que la norme du résidu décroit (de 10^{-3} à 10^{-5} approximativement). Pour le preconditionneur de Jacobi, la convergence est aussi plus rapide. Cependant, les itérés 3, 4 et 5 font augmenter la norme du résidu ce qui n'est pas souhaitable.