



Rapport Projet Calcul Scientifique et Analyse de Données TP2 - Calcul des Couples Propres

Faical TOUBALI HADAOUI
Younes EL BOUZEKRAOUI

Département Sciences du Numérique - Première année
2019-2020

Table des matières

| | | |
|----------|--|----------|
| 1 | Partie 1 : Limitations of the power method | 4 |
| 1.1 | Question 1 | 4 |
| 1.2 | Question 2 | 4 |
| 2 | Partie 2 : Extending the power method to compute dominant eigenspace vectors | 4 |
| 2.1 | subspace iter v0 : a basic method to compute a dominant eigenspace | 4 |
| 2.1.1 | Question 3 : Orthonormalisation | 4 |
| 2.1.2 | Question 4 : Rayleigh quotient | 6 |
| 2.1.3 | Question 5 : Implantation iter v0 | 6 |
| 2.2 | Subspace iter v1 : improved version making use of Raleigh-Ritz projection | 6 |
| 2.2.1 | Question 6 : | 6 |
| 3 | Partie 3 : Subspace iter v2 and subspace iter v3 : toward an efficient solver | 6 |
| 3.1 | Question 7 : | 6 |
| 3.2 | Question 8 : Block approach (subspace iter v2) | 7 |
| 3.3 | Deflation method (subspace iter v3) | 7 |
| 3.3.1 | Question 9 : | 7 |
| 3.3.2 | Question 10 : | 8 |
| 3.3.3 | Question 11 : | 8 |
| 3.3.4 | Question 12 : | 8 |
| 3.3.5 | Question 13 : | 10 |
| 3.3.6 | Question 14 : | 12 |

Table des figures

| | | |
|----|--|----|
| 1 | Time for the Lapack subroutine dsyev | 4 |
| 2 | Time for the deflated power method | 4 |
| 3 | Vecteurs propres reçus par la fonction eig | 5 |
| 4 | Matrice V obtenue par l'algorithme 2 | 5 |
| 5 | Algorithme sans orthonormalisation | 5 |
| 6 | Matrice V (sans orthonormalisation) | 6 |
| 7 | Temps mis par iter v1 | 7 |
| 8 | Temps mis par iter v2 | 7 |
| 9 | Accuracy of vectors | 7 |
| 10 | Temps mis par iter v3 | 8 |
| 11 | Temps mis par iter v2 | 8 |
| 12 | iter v2 avec p=1 | 9 |
| 13 | iter v2 avec p=5 | 9 |
| 14 | valeurs propres de la matrice de type 1 calculé par iter v3 | 10 |
| 15 | valeurs propres de la matrice de type 2 calculé par iter v3 | 10 |
| 16 | valeurs propres de la matrice de type 3 calculé par iter v3 | 10 |
| 17 | valeurs propres de la matrice de type 4 calculé par iter v3 | 11 |
| 18 | script pour affiche la distribution des valeurs propres en matlab | 11 |
| 19 | Distribution des valeurs propres pour les différents types de matrice | 11 |
| 20 | comparaison des différentes méthodes pou une matrice de type 2 de taille 500 | 12 |
| 21 | comparaison des différentes méthodes pou une matrice de type 3 de taille 500 | 13 |
| 22 | comparaison des différentes méthodes pou une matrice de type 2 de taille 50 | 14 |
| 23 | Temps pris par la methode eig | 15 |
| 24 | Temps pris par la methode iter v0 | 15 |
| 25 | Temps pris par la methode iter v1 | 15 |

| | | |
|----|---|----|
| 26 | Temps pris par la methode iter v2 | 15 |
| 27 | Distribution des valeurs propres pour chaque algorithme | 15 |

1 Partie 1 : Limitations of the power method

1.1 Question 1

Pour une matrice de type 2 et de taille 1000 et on compare le temps mis par les deux methodes Deflated Power Method et Lapack Subroutine dsyev :

```
=====
Time =    0.10400000214576721
=====
```

FIGURE 1 – Time for the Lapack subroutine dsyev

```
Calling the basic deflated power method
End of the basic deflated power method
maxit reached : ERROR
hints: increase maxit or decrease eps in main.f90
An error was found. IERR= -3

=====
Time =    8.5769996643066406
=====
```

FIGURE 2 – Time for the deflated power method

Commentaire :

On remarque que la méthode deflated power method est moins performante en terme de temps en comparaison avec Lapack subroutine dsyev, en effet, le temps pris par la methode deflated power method est superieur 85 fois par rapport a la methode Lapack subroutine dsyev.

1.2 Question 2

L'inconvénient de la Deflated Power Methode est le fait qu'il y a beaucoup de multiplications matricielle $A*v$ a chaque itération, ce que la rend moins efficace en terme de temps.

2 Partie 2 : Extending the power method to compute dominant eigenspace vectors

2.1 subspace iter v0 : a basic method to compute a dominant eigenspace

2.1.1 Question 3 : Orthonormalisation

On constate que les deux algorithmes 1 et 2 sont très similaires sauf que dans le premier on agit sur un vecteur v et dans le deuxième on agit sur une matrice V de vecteurs v , et puisque v tend dans l'algorithme 1 vers un vecteur propre, on peut conjecturer que la matrice V va tendre vers une matrice contenant les vecteurs propres de notre matrice initiale.

On adapte le script de la puissance itérée à l'algorithme 2 et on test pour $n=5$ et $m=5$. On compare le contenu de la matrice V avec les vecteurs propre générés par la fonction eig de matlab qui rend une liste des vecteurs propres et leurs valeurs propres associés dans une matrice associé :

```
K>> [a,b]=eig(M);
K>> a
```

a =

| | | | | |
|---------|---------|---------|---------|---------|
| -0.5358 | 0.1129 | -0.0967 | 0.7059 | 0.4387 |
| 0.6026 | -0.3923 | -0.5716 | 0.2532 | 0.3035 |
| -0.0880 | -0.8164 | 0.4748 | 0.2498 | -0.1946 |
| -0.2186 | -0.2708 | 0.0959 | -0.5688 | 0.7390 |
| 0.5424 | 0.3057 | 0.6552 | 0.2273 | 0.3624 |




FIGURE 3 – Vecteurs propres reçus par la fonction eig

v =

| | | | | |
|---------|---------|---------|---------|---------|
| 0.4387 | 0.7059 | -0.0967 | -0.1129 | -0.5358 |
| 0.3035 | 0.2532 | -0.5716 | 0.3923 | 0.6026 |
| -0.1946 | 0.2498 | 0.4748 | 0.8164 | -0.0880 |
| 0.7390 | -0.5688 | 0.0959 | 0.2708 | -0.2186 |
| 0.3624 | 0.2273 | 0.6552 | -0.3057 | 0.5424 |




FIGURE 4 – Matrice V obtenue par l'algorithme 2

On remarque que dans l'algorithme 2 le vecteur V converge vers les vecteurs propres de la matrice initiale mais d'une façon non ordonnés.

Alors notre conjecture est correcte et le vecteur V tend vers une matrice contenant les vecteurs propres de notre matrice initiale.

En supprimant l'orthonormalisation des colonnes de Y a la fin de chaque itération, on ne trouve plus les vecteurs propres de la matrice initiale et le vecteur V ne converge pas vers la matrice des vecteurs propres.

```
while(~cv)
    k = k + 1;
    Y=M*V;
    H=V'*M*V;
    err1 = (norm(M*V-V*H,'fro'))/(norm(M,'fro'));
    %Orthonormalisation
    %V=orth(Y);
    V=Y
    cv = (err1 < eps) | (k > kmax);
end
```

FIGURE 5 – Algorithme sans orthonormalisation

```

V =
-76.9374    80.8896     0.3724   -38.3365   -23.4166
193.4534   -82.9213    77.4739   -55.8243    23.9139
184.9526  -208.7936    18.9183   -77.3765   -38.5052
 73.1855    58.1241   -1.0424   -91.1740   -22.6558
  4.7083   -24.6889   104.5368   -15.2714   -21.6207

```

FIGURE 6 – Matrice V (sans orthonormalisation)

2.1.2 Question 4 : Rayleigh quotient

On a $A \in R^{n \times n}$ et $V \in R^{n \times m}$ avec $m \leq n$
 où m le nombre de couples propres qu'on veut calculer
 alors $H \in R^{m \times m}$.

Alors le calcul de decomposition spectrale de H sera moins couteux et donc plus rapide que celui de A.

2.1.3 Question 5 : Implantation iter v0

Voir code Fortran iter v0

2.2 Subspace iter v1 : improved version making use of Raleigh-Ritz projection

2.2.1 Question 6 :

Le code fournit iter v1 identifie toutes les étapes de l'algorithme 4.

3 Partie 3 : Subspace iter v2 and subspace iter v3 : toward an efficient solver

3.1 Question 7 :

Produit pour calculer les coefficients de $A \times B$:
 avec $A \in R^{n,m}$ et $B \in R^{n',m'}$

Produit pour calculer les coefficient de $A \times B$:
 $A \in R^{n,m}$ $B \in R^{m,k}$

On a n*k coefficient

Pour calculer chaque coefficient il faut $m \times m = m^2$ multiplications et m - 1 additions

Donc pour calculer $A \times B$ il faut : $(n \times k) \times (m^2 + m - 1)$

Dans ce cas on a n = m = k

alors $m^2 \times (m^2 + m - 1)$

Or pour calculer A^p il faut p - 1 itérations alors le nombre d'opérations totale est de
 $(p - 1) \times (m^2 \times (m^2 + m - 1))$

A^p étant déjà claculé :

Le nombre d'operations pour calculer $A^p \times V(V(n, m))$ est :

$$(n \times m) \times (n^2 + n - 1)$$

3.2 Question 8 : Block approach (subspace iter v2)

Après l'implantation de l'algorithme de iter v2 on fait un test pour comparer les performances de iter v1 et iter v2.

On calcule les 20 premiers couples propres pour une matrice de type 2 de taille 500.

```
=====
Time =      2.6789999008178711
=====
```

FIGURE 7 – Temps mis par iter v1

```
=====
Time =      2.5820000171661377
=====
```

FIGURE 8 – Temps mis par iter v2

On peut conclure que la méthode iter v2 est plus performante en terme de temps que iter v1 mais d'une différence très petite.

3.3 Deflation method (subspace iter v3)

3.3.1 Question 9 :

En lançant l'algorithme iter v1 avec l'option disp 3 on peut voir les precisions calculées (acc) au cours de l'algorithme et la fin pour chaque couple propre.

```
10 First eigenvalues
=====
Eigenvalue 1: 1.000 accuracy : 0.778E-15 number of iterations : 51
Eigenvalue 2: 0.997 accuracy : 0.803E-15 number of iterations : 52
Eigenvalue 3: 0.903 accuracy : 0.994E-15 number of iterations : 59
Eigenvalue 4: 0.891 accuracy : 0.697E-15 number of iterations : 61
Eigenvalue 5: 0.878 accuracy : 0.761E-15 number of iterations : 64
Eigenvalue 6: 0.781 accuracy : 0.828E-15 number of iterations : 77
Eigenvalue 7: 0.702 accuracy : 0.837E-15 number of iterations : 112
Eigenvalue 8: 0.653 accuracy : 0.980E-15 number of iterations : 135
Eigenvalue 9: 0.644 accuracy : 0.965E-15 number of iterations : 137
Eigenvalue 10: 0.553 accuracy : 0.970E-15 number of iterations : 538

||A*V_i - Lambda_i*V_i||/||A||
=====
Eigenvalue 1: 0.278E-15
Eigenvalue 2: 0.334E-15
Eigenvalue 3: 0.198E-15
Eigenvalue 4: 0.444E-15
Eigenvalue 5: 0.514E-15
Eigenvalue 6: 0.208E-15
Eigenvalue 7: 0.330E-15
Eigenvalue 8: 0.221E-15
Eigenvalue 9: 0.208E-15
Eigenvalue 10: 0.976E-15
```

FIGURE 9 – Accuracy of vectors

On remarque que la precision (accuracy) des couples propres calculé au cours de l'algorithme est différente à celle calculé la fin, cela revient au fait que lorsque les premières colonnes de V convergent on n'arrête pas le calcul mais on continue de faire les opérations de l'algorithme sur ces mêmes vecteurs même si ils ont déjà convergé.

Alors forcément le calcul de la precision (accuracy) sera différent pour les vecteurs propres au cours de l'algorithme et à la fin.

3.3.2 Question 10 :

En ce qui concerne la methode subspace iter v3 on peut anticiper que la precision (accuracy) des vecteurs propres serait la même au cours de l'algorithme et à la fin car iter v3 consiste à ne plus toucher les vecteurs qui ont convergé et ne pas faire les opérations de l'algorithme sur ces derniers , Ainsi la precision(accuracy) d'un vecteur propres ne changera pas après vers la fin de l'algorithme.

3.3.3 Question 11 :

On implémente l'algorithme iter v3 sur fortran en modifiant dans le script de iter v2 l'étape d'orthonormalisation, et en remplaçant la routine gram schmidt par routine gram schmidt block, et en n'agissant plus sur les vecteurs de la matrice V qui ont déjà convergé .

On teste les performances de ce nouveau algorithme pour une matrice de type 2 et de taille 500, en essayant d'arriver à un pourcentage 50

```
yelbouze2@z6po:~/annee_1/Calcul scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$ ./main -imat 2 -n 500 -v 3 -m 20 -per 0.5 -disp 0
Calling the subspace iteration method v3
End the subspace iteration method v3
percentage 0.50 reached with 18 eigenvalues
n_ev = 18
=====
Time = 0.69999998887907104
=====
```

FIGURE 10 – Temps mis par iter v3

```
Calling the subspace iteration method v2
End the subspace iteration method v2
percentage 0.50 reached with 18 eigenvalues
n_ev = 18
=====
Time = 0.79400002956390381
=====
```

FIGURE 11 – Temps mis par iter v2

3.3.4 Question 12 :

En augmentant le nombre p on remarque que le temps d'exécution diminue et les couples propres sont trouvés plus rapidement avec un nombre d'itération réduit par rapport au cas où p=1. On fait une comparaison pour iter v2 entre p=1 et p=5


```

Calling the subspace iteration method v2

End the subspace iteration method v2
percentage 0.70 NOT reached with 20 eigenvalues
n_ev = 20

20 First eigenvalues
=====
Eigenvalue 1: 1.000 accuracy : 0.767E-15 number of iterations : 40
Eigenvalue 2: 0.997 accuracy : 0.374E-15 number of iterations : 40
Eigenvalue 3: 0.903 accuracy : 0.748E-15 number of iterations : 45
Eigenvalue 4: 0.891 accuracy : 0.701E-15 number of iterations : 45
Eigenvalue 5: 0.878 accuracy : 0.628E-15 number of iterations : 46
Eigenvalue 6: 0.781 accuracy : 0.829E-15 number of iterations : 49
Eigenvalue 7: 0.702 accuracy : 0.955E-15 number of iterations : 64
Eigenvalue 8: 0.653 accuracy : 0.611E-15 number of iterations : 72
Eigenvalue 9: 0.644 accuracy : 0.432E-15 number of iterations : 72
Eigenvalue 10: 0.553 accuracy : 0.990E-15 number of iterations : 112
Eigenvalue 11: 0.524 accuracy : 0.999E-15 number of iterations : 136
Eigenvalue 12: 0.517 accuracy : 0.763E-15 number of iterations : 136
Eigenvalue 13: 0.477 accuracy : 0.950E-15 number of iterations : 207
Eigenvalue 14: 0.474 accuracy : 0.926E-15 number of iterations : 236
Eigenvalue 15: 0.470 accuracy : 0.688E-15 number of iterations : 236
Eigenvalue 16: 0.460 accuracy : 0.915E-15 number of iterations : 291
Eigenvalue 17: 0.458 accuracy : 0.921E-15 number of iterations : 298
Eigenvalue 18: 0.456 accuracy : 0.995E-15 number of iterations : 302
Eigenvalue 19: 0.451 accuracy : 0.996E-15 number of iterations : 346
Eigenvalue 20: 0.422 accuracy : 0.998E-15 number of iterations : 1173

=====
Time = 2.8829998970031738
=====

```

FIGURE 12 – iter v2 avec p=1

```

Calling the subspace iteration method v2

End the subspace iteration method v2
percentage 0.70 NOT reached with 20 eigenvalues
n_ev = 20

20 First eigenvalues
=====
Eigenvalue 1: 1.000 accuracy : 0.233E-15 number of iterations : 8
Eigenvalue 2: 0.997 accuracy : 0.398E-15 number of iterations : 8
Eigenvalue 3: 0.903 accuracy : 0.365E-15 number of iterations : 9
Eigenvalue 4: 0.891 accuracy : 0.591E-15 number of iterations : 9
Eigenvalue 5: 0.878 accuracy : 0.518E-15 number of iterations : 9
Eigenvalue 6: 0.781 accuracy : 0.221E-15 number of iterations : 11
Eigenvalue 7: 0.702 accuracy : 0.982E-15 number of iterations : 12
Eigenvalue 8: 0.653 accuracy : 0.836E-15 number of iterations : 14
Eigenvalue 9: 0.644 accuracy : 0.325E-15 number of iterations : 15
Eigenvalue 10: 0.553 accuracy : 0.361E-15 number of iterations : 22
Eigenvalue 11: 0.524 accuracy : 0.836E-15 number of iterations : 25
Eigenvalue 12: 0.517 accuracy : 0.432E-15 number of iterations : 27
Eigenvalue 13: 0.477 accuracy : 0.991E-15 number of iterations : 40
Eigenvalue 14: 0.474 accuracy : 0.517E-15 number of iterations : 45
Eigenvalue 15: 0.470 accuracy : 0.725E-15 number of iterations : 45
Eigenvalue 16: 0.460 accuracy : 0.587E-15 number of iterations : 51
Eigenvalue 17: 0.458 accuracy : 0.731E-15 number of iterations : 56
Eigenvalue 18: 0.456 accuracy : 0.709E-15 number of iterations : 56
Eigenvalue 19: 0.451 accuracy : 0.964E-15 number of iterations : 64
Eigenvalue 20: 0.422 accuracy : 0.894E-15 number of iterations : 215

=====
Time = 1.3760000467300415
=====

```

FIGURE 13 – iter v2 avec p=5

Lorsque on augmente p, on économise en terme du nombre d'itération nécessaires, en effet, au

lieu de répéter l'itération $V=A*V$ n fois ce qui est équivalent a calculer $V = A^n * v$ pour $p=1$ on ne la repete que $n/2$ fois lorsque $p=2$ car on va faire l'opération $V = A^2 * V$ $n/2$ fois .

3.3.5 Question 13 :

Pour visualiser les différence entre les 4 type de matrice(imat 1,imat 2,imat 3,imat 4), on trace des figures affichant les distribution des valeurs propres pour ces différents types de matrices. Les matrice utilisées sont de tailles 200*200 et l'algorithme utilisé est subspace iter v3. On a calculé les valeurs propres avec fortran et on les ai mis dans des listes sur matlab pour afficher leurs distributions.

```

yelbouze2@z6po:~/annee_1/Calcul scientifique/Projet_TP2_Fournitures/code_fournl/fortran$ ./main -lnat 1 -n 200 -v 3 -n 10 -disp 1
Here are the first 10 eigenvalues:  200.000  199.000  198.000  197.000  196.000  195.000  194.000  193.000  192.000  191.000
Calling the subspace iteration method v3
End the subspace iteration method v3
percentage 0.70 NOT reached with 10 eigenvalues
n_ev = 10

10 First eigenvalues
=====
Eigenvalue 1: 200.000 accuracy : 0.993E-15 number of iterations : 524
Eigenvalue 2: 199.000 accuracy : 0.983E-15 number of iterations : 630
Eigenvalue 3: 198.000 accuracy : 0.998E-15 number of iterations : 695
Eigenvalue 4: 197.000 accuracy : 0.977E-15 number of iterations : 818
Eigenvalue 5: 196.000 accuracy : 0.984E-15 number of iterations : 911
Eigenvalue 6: 195.000 accuracy : 0.976E-15 number of iterations : 1009
Eigenvalue 7: 194.000 accuracy : 0.100E-14 number of iterations : 1306
Eigenvalue 8: 193.000 accuracy : 0.999E-15 number of iterations : 1786
Eigenvalue 9: 192.000 accuracy : 0.996E-15 number of iterations : 2643
Eigenvalue 10: 191.000 accuracy : 0.986E-15 number of iterations : 5052
=====
Time = 0.54900002479553223
=====

```

FIGURE 14 – valeurs propres de la matrice de type 1 calculé par iter v3

```

yelbouze2@z6po:~/annee_1/Calcul scientifique/Projet_TP2_Fournitures/code_fournl/fortran$ ./main -lnat 2 -n 200 -v 3 -n 10 -disp 1
Here are the first 10 eigenvalues:  1.000  0.744  0.597  0.589  0.544  0.540  0.536  0.481  0.419  0.370
Calling the subspace iteration method v3
End the subspace iteration method v3
percentage 0.70 NOT reached with 10 eigenvalues
n_ev = 10

10 First eigenvalues
=====
Eigenvalue 1: 1.000 accuracy : 0.685E-15 number of iterations : 35
Eigenvalue 2: 0.744 accuracy : 0.718E-15 number of iterations : 46
Eigenvalue 3: 0.597 accuracy : 0.684E-15 number of iterations : 68
Eigenvalue 4: 0.589 accuracy : 0.737E-15 number of iterations : 69
Eigenvalue 5: 0.544 accuracy : 0.679E-15 number of iterations : 75
Eigenvalue 6: 0.540 accuracy : 0.962E-15 number of iterations : 82
Eigenvalue 7: 0.536 accuracy : 0.902E-15 number of iterations : 82
Eigenvalue 8: 0.481 accuracy : 0.984E-15 number of iterations : 103
Eigenvalue 9: 0.419 accuracy : 0.911E-15 number of iterations : 199
Eigenvalue 10: 0.370 accuracy : 0.995E-15 number of iterations : 1056
=====
Time = 8.6999997496604919E-002
=====

```

FIGURE 15 – valeurs propres de la matrice de type 2 calculé par iter v3

```

yelbouze2@z6po:~/annee_1/Calcul scientifique/Projet_TP2_Fournitures/code_fournl/fortran$ ./main -lnat 3 -n 200 -v 3 -n 10 -disp 1
Here are the first 10 eigenvalues:  100.000  94.379  89.074  84.067  79.341  74.881  70.672  66.699  62.950  59.411
Calling the subspace iteration method v3
End the subspace iteration method v3
percentage 0.70 NOT reached with 10 eigenvalues
n_ev = 10

10 First eigenvalues
=====
Eigenvalue 1: 100.000 accuracy : 0.937E-15 number of iterations : 56
Eigenvalue 2: 94.379 accuracy : 0.765E-15 number of iterations : 61
Eigenvalue 3: 89.074 accuracy : 0.682E-15 number of iterations : 69
Eigenvalue 4: 84.067 accuracy : 0.880E-15 number of iterations : 82
Eigenvalue 5: 79.341 accuracy : 0.911E-15 number of iterations : 95
Eigenvalue 6: 74.881 accuracy : 0.806E-15 number of iterations : 112
Eigenvalue 7: 70.672 accuracy : 0.926E-15 number of iterations : 140
Eigenvalue 8: 66.699 accuracy : 0.924E-15 number of iterations : 173
Eigenvalue 9: 62.950 accuracy : 0.914E-15 number of iterations : 273
Eigenvalue 10: 59.411 accuracy : 0.995E-15 number of iterations : 467
=====
Time = 5.4000001400709152E-002
=====

```

FIGURE 16 – valeurs propres de la matrice de type 3 calculé par iter v3

```

yelbouze2@r6po:~/annee_1/calcul_scientifique/Projet_TP2_Fournitures/code_fourni/fortran5 ./main -lmat 4 -n 200 -v 3 -m 10 -disp 1
Here are the first 10 eigenvalues: 1000.000 995.025 990.050 985.075 980.101 975.126 970.151 965.176 960.201 955.226
Calling the subspace iteration method v3
End the subspace iteration method v3
percentage 0.78 NOT reached with 10 eigenvalues
n_ev = 10

10 First eigenvalues
=====
Eigenvalue 1: 1000.000 accuracy : 0.978E-15 number of iterations : 633
Eigenvalue 2: 995.025 accuracy : 0.973E-15 number of iterations : 692
Eigenvalue 3: 990.050 accuracy : 0.966E-15 number of iterations : 763
Eigenvalue 4: 985.075 accuracy : 0.991E-15 number of iterations : 882
Eigenvalue 5: 980.101 accuracy : 0.991E-15 number of iterations : 997
Eigenvalue 6: 975.126 accuracy : 0.992E-15 number of iterations : 1228
Eigenvalue 7: 970.151 accuracy : 0.985E-15 number of iterations : 1373
Eigenvalue 8: 965.176 accuracy : 0.996E-15 number of iterations : 1958
Eigenvalue 9: 960.201 accuracy : 0.994E-15 number of iterations : 2878
Eigenvalue 10: 955.226 accuracy : 0.998E-15 number of iterations : 5899

=====
Time = 0.62199997901916584
=====

```

FIGURE 17 – valeurs propres de la matrice de type 4 calculé par iter v3

```

%Q13 - Distribution des valeurs propres pour les différentes matrices
% dimensions des matrices 200x200
% algorithme utilisé : subspace_iter_v3
%Nombre des valeurs propres à afficher ( Valeurs prises à partir du
%terminal en lançant les implantations des algo en Fortran )
a = 10;
T1 = [200 199 198 197 196 195 194 193 192 191];
T2 = [1.000 0.744 0.597 0.589 0.544 0.540 0.536 0.481 0.419 0.370];
T3 = [100.000 94.379 89.074 84.067 79.341 74.881 70.672 66.699 62.950 59.411];
T4 = [1000 995.025 990.050 985.075 980.101 975.126 970.151 965.176 960.201 955.226];

figure(2),clf

subplot(4,1,1)
p1=plot(T1,zeros(1,a),'k*');
grid on;
title('Distribution des valeurs propres pour les différentes matrices');
subplot(4,1,2)
p2=plot(T2,zeros(1,a),'r*');
grid on;
subplot(4,1,3)
p3=plot(T3,zeros(1,a),'g*');
grid on;
subplot(4,1,4)
p4=plot(T4,zeros(1,a),'b*');
grid on;
legend([p1;p2;p3;p4],'Matrice type 1','Matrice type 2','Matrice type 3','Matrice type 4');

```

FIGURE 18 – script pour affiche la distribution des valeurs propres en matlab

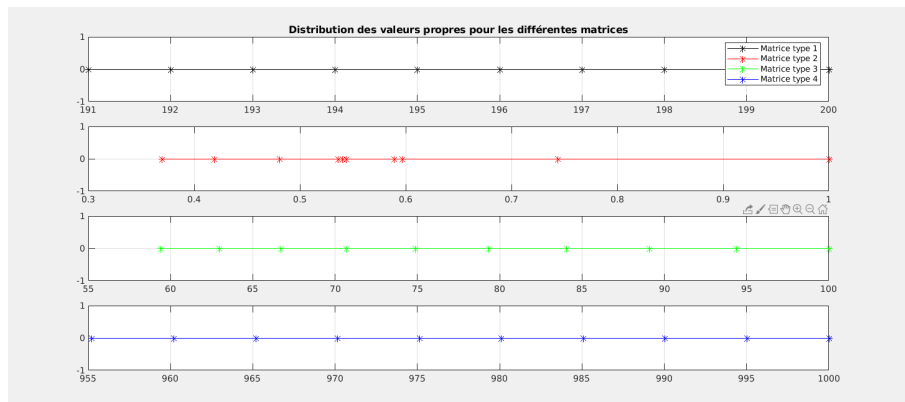


FIGURE 19 – Distribution des valeurs propres pour les différents types de matrice

Remarques :

Pour les matrices de type 1, la distance entre les valeurs propres est constante de valeur 1, tandis que cette distance n'est plus constante pour les autres type de matrices.

On remarque aussi que la distribution des valeurs propres pour la matrice de type 2 est mal ordonnée par rapport aux autres type matrices.

Concernant le type 4 des matrices on remarque que leurs valeurs propres sont très grandes en comparaison avec les autres type de matrices.

3.3.6 Question 14 :

Comparaison en Fortran : On compare d'abord les différentes méthodes implantées pour une matrice de type 2 de taille 500 : On remarque que DSYEV est la plus performante et pour les autres je les cite par ordre décroissant de performance en terme de temps : iter v3 puis iter v2 puis iter v1 puis iter v0.

```
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 500 -v 10 -m 20 -per 0.5 -disp 0

Calling LAPACK DSYEV on A
End of LAPACK DSYEV

=====
Time = 0.10400000214576721
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 500 -v 0 -m 20 -per 0.5 -disp 0

Calling the basic subspace iteration method v0
IT: 1072 -- Accuracy is: 9.78E-16
End of the basic subspace iteration method

20 Eigenvalues computed in 1072 iterations with a residual of 0.978E-15

=====
Time = 1.5729999542236328
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 500 -v 1 -m 20 -per 0.5 -disp 0

Calling the subspace iteration method v1

End the subspace iteration method v1

percentage 0.50 reached with 18 eigenvalues
n_ev = 18

=====
Time = 0.73000001907348633
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 500 -v 2 -m 20 -per 0.5 -disp 0

Calling the subspace iteration method v2

End the subspace iteration method v2

percentage 0.50 reached with 18 eigenvalues
n_ev = 18

=====
Time = 0.72200000286102295
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 500 -v 3 -m 20 -per 0.5 -disp 0

Calling the subspace iteration method v3

End the subspace iteration method v3

percentage 0.50 reached with 18 eigenvalues
n_ev = 18

=====
Time = 0.40700000524520874
=====
```

FIGURE 20 – comparaison des différentes méthodes pour une matrice de type 2 de taille 500

Ensuite on les compare pour une matrice de type 3 de taille 500 : DSYEV est toujours la plus performante, suivie de iter v3 mais dans ce cas iter v1 et plus performante que iter v2 .

```
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 3 -n 500 -v 10 -m 20 -disp 0

Calling LAPACK DSYEV on A
End of LAPACK DSYEV

=====
Time = 0.14399999380111694
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 3 -n 500 -v 0 -m 20 -disp 0

Calling the basic subspace iteration method v0
IT: 1411 -- Accuracy is: 9.85E-16
End of the basic subspace iteration method

20 Eigenvalues computed in 1411 iterations with a residual of 0.985E-15

=====
Time = 2.0590000152587891
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 3 -n 500 -v 1 -m 20 -disp 0

Calling the subspace iteration method v1

End the subspace iteration method v1
percentage 0.70 NOT reached with 20 eigenvalues
n_ev = 20

=====
Time = 3.1080000400543213
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 3 -n 500 -v 2 -m 20 -disp 0

Calling the subspace iteration method v2

End the subspace iteration method v2
percentage 0.70 NOT reached with 20 eigenvalues
n_ev = 20

=====
Time = 3.5529999732971191
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 3 -n 500 -v 3 -m 20 -disp 0

Calling the subspace iteration method v3

End the subspace iteration method v3
percentage 0.70 NOT reached with 20 eigenvalues
n_ev = 20

=====
Time = 0.87999999523162842
=====
```

FIGURE 21 – comparaison des différentes méthodes pour une matrice de type 3 de taille 500

Finalement on compare les méthodes pour une petite matrice de taille 50 de type 2 : Dans ce cas iter v2 est plus performante que DSYEV , iter v1, iter v3 et iter v0.

```

yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 50 -v 10 -m 20 -disp 0

Calling LAPACK DSYEV on A
End of LAPACK DSYEV

=====
Time = 2.0000000949949026E-003
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 50 -v 0 -m 20 -disp 0

Calling the basic subspace iteration method v0
IT: 235 -- Accuracy is: 9.30E-16
End of the basic subspace iteration method

20 Eigenvalues computed in 235 iterations with a residual of 0.930E-15

=====
Time = 1.7999999225139618E-002
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 50 -v 1 -m 20 -disp 0

Calling the subspace iteration method v1
End the subspace iteration method v1

percentage 0.70 reached with 5 eigenvalues
n_ev = 5

=====
Time = 3.0000000260770321E-003
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 50 -v 2 -m 20 -disp 0

Calling the subspace iteration method v2
End the subspace iteration method v2

percentage 0.70 reached with 5 eigenvalues
n_ev = 5

=====
Time = 1.0000000474974513E-003
=====
yelbouze2@z6po:~/annee_1/Calcul_scientifique/Projet_TP2_Fournitures/code_fourni/Fortran$
./main -imat 2 -n 50 -v 3 -m 20 -disp 0

Calling the subspace iteration method v3
End the subspace iteration method v3

percentage 0.70 reached with 5 eigenvalues
n_ev = 5

=====
Time = 2.0000000949949026E-003
=====

```

FIGURE 22 – comparaison des différentes méthodes pour une matrice de type 2 de taille 50

Comparaison en Matlab : Pour une matrice donnée dans le fichier main de matlab, matrice rectangulaire aléatoire de A taille 1000*50, on compare le temps pris par chaque algorithme pour calculer les couples propres.

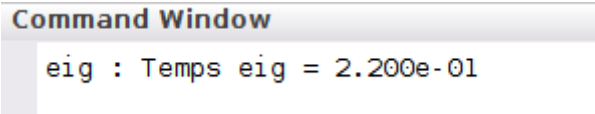


FIGURE 23 – Temps pris par la methode eig

```

iter0 : Temps v0 = 2.740e+00
iter0 : Qualité de la solution valeurs propres / eig = 1.985e-14
iter0 : Qualité de la solution couple propres = [7.634e-07 - 2.881e-05]

```

FIGURE 24 – Temps pris par la methode iter v0

```

iter1 : Temps v1 = 6.700e-01
iter1 : Qualité de la solution valeurs propres / eig = 4.954e-15
iter1 : Qualité de la solution couple propres = [5.213e-15 - 4.962e-08]

```

FIGURE 25 – Temps pris par la methode iter v1

```

iter2 : Temps v2 = 3.200e-01
iter2 : Qualité de la solution valeurs propres / eig = 2.484e-15
iter2 : Qualité de la solution couple propres = [8.472e-16 - 1.828e-08]

```

FIGURE 26 – Temps pris par la methode iter v2

On remarque que les temps des algorithmes sous matlab sont compatibles avec ceux calculés par fortran, à savoir l'algorithme subspace iter v2 est plus rapide que subspace iter v1 qui est à son tour plus rapide que subspace iter v0.

Pour visualiser la précision apporté par chaque algorithme pour le calcul des éléments propres, on trace un graphe contenant la distribution des valeurs propres.

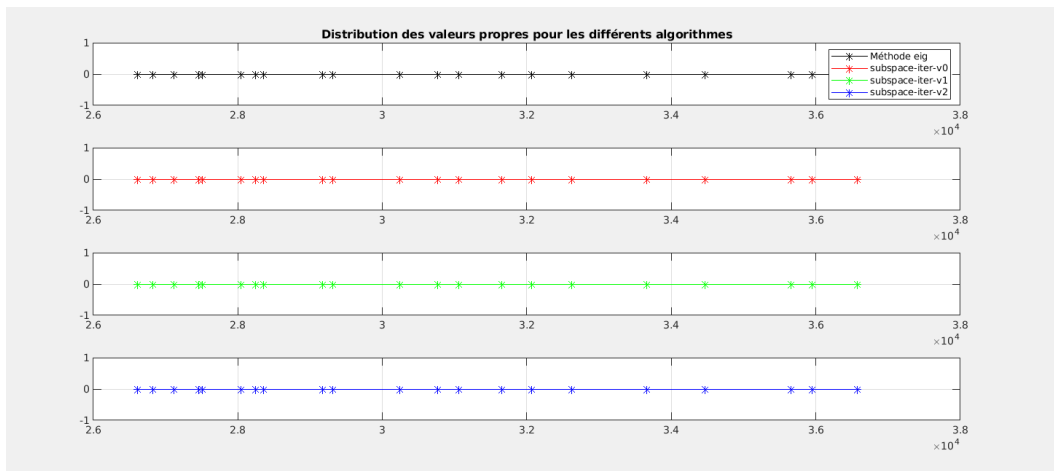


FIGURE 27 – Distribution des valeurs propres pour chaque algorithme

Effectivement on ne remarque pas une grande différence entre les valeurs propres en terme de precision.