TP-Projet 2 : Calcul des Couples Propres

Nom des auteurs
Cazes Noa
James Christopher
Martin Cédric

Département Sciences du Numérique - Première année
2019-2020

Table des matières

1 Introduction **3**

2 Limites de la méthode de puissance itérée **3**

3 Extension de la méthode la puissance itérée pour calculer les couples propres principaux **3**

4 L'approche par blocs et la méthode de déflation : vers une solution plus efficace **5**

5 Conclusion **8**

Table des figures

1 Distribution des valeurs propres de la matrice 1 6
2 Distribution des valeurs propres de la matrice 2 6
3 Distribution des valeurs propres de la matrice 3 7
4 Distribution des valeurs propres de la matrice 4 7

1 Introduction

De la première partie de ce projet, nous avons pu retenir que pour réduire les dimensions d'une matrice à l'aide de l'ACP nous avions seulement besoin des couples propres jugés essentiels, dans le sens où ils permettent de restaurer un maximum de l'information initialement contenue dans la matrice en question.

L'objectif de cette deuxième partie de projet est alors de comparer différents algorithmes permettant d'obtenir ces couples propres, d'où la détermination du plus rapide.

2 Limites de la méthode de puissance itérée

Ici, l'étude se porte sur l'algorithme de puissance itérée avec déflation.

Question 1

On observe que l'algorithme de la puissance itérée avec déflation a un temps de convergence beaucoup plus long que celui de la fonction *dsyev*.

Question 2

Le principal inconvénient de cette méhode est l'assignation $\beta_{old} = \beta$.

3 Extension de la méthode la puissance itérée pour calculer les couples propres principaux

Question 3

La matrice V converge vers la matrice des vecteurs propres de la matrice A exprimée dans la base dans laquelle H est diagonale - ou la matrice des vecteurs propres de H (matrice de passage entre la base des vecteurs propres de H et la base canonique).

Ça a complètement été vérifié par la pratique. En effet, on récupère les vecteurs propres et valeurs propres de H et A afin de vérifier que $VX =$ vp de A.

Question 4

Le fait que l'algorithme 2 calcule l'entière décomposition spectrale de H n'est pas un problème car H est de dimension moindre (20x20).

Question 5

Voir le code correspondant à la v0.

Question 6

Par rapport à l'algorithme donné, on a pu identifier les étapes suivantes :

$k = k + 1$

$Y = AV$

```
!! A. Compute  y = a*v
call dgemm('n', 'n', n, m, n, done, a, n, v, n, dzero, y, n)
```

Orthonormalisation des colonnes de Y.

```
!! B. Orthonormalisation
call gram_schmidt(y, n, m, v)
```

Rayleigh-Ritz projection appliqué sur A et V.

Calculer le quotient de Rayleigh $H = V^T AV$.

```
!! C. Rayleigh-Ritz projection
!!   1. H = V^T A V
!!      Y = A V
call dgemm('n','n', n, m, n, done, a, n, v, n, dzero, y, n)
!!      H = V'*Y
call dgemm('t', 'n', m, m, n, done, v, n, y, n, dzero, h, m)
```

Calculer la décomposition spectrale $X\Lambda_{out}X^T = H$, où les valeurs propres de H ($diag(\Lambda_{out})$) sont arrangées dans l'odre décroissant.

```
!!   2. Spectral decomposition
call dsyev('v', 'u', m, h, m, w_aux, work, lwork, ierr)
if( ierr .ne.0 )then
  write(*,'("Error in dsyev")')
  ierr = -4
  goto 999
end if

!! Sort in the decreasing order
!! (we suppose that all the eigen values are positve)
do i = 1, m
  t(i) = w_aux(m-i+1)
  x(:, i)  = h(:, m-i+1)
end do
```

Compute $V_{out} = VX$.

```
y = v
call dgemm('n', 'n', n, m, m, done, y, n, x, m, dzero, v, n)
```

Conserver les couples propres qui ont convergé et ont atteint un pourcentage donné.

```
!! D. Convergence analysis step
conv = 0
i = n_ev + 1
!! the larger eigenvalue will converge more swiftly than
!! those corresponding to the smaller eigenvalue.
!! for this reason, we test the convergence in the order
!! i=1,2,.. and stop with the first one to fail the test
ok = .false.
do while(.not. ok)
  if( i .gt. m) then
    ok = .true.
  else
    !!compute acc=norm(a*v(:,i) - v(:,i)*t(i),2)/lambda;
    !!--compute aux_acc=a*v(:,i) - v(:,i)*t(i)
    !!--compute acc=||aux_acc||/||a||
    aux_acc = v(:,i)
    beta = -t(i)
    call dgemv('n', n, n, done, a, n, v(1,i), ione, beta, aux_acc, ione)
    acc = sqrt(ddot(n, aux_acc, ione, aux_acc, ione))/normF_A
    ! write(*,*) i, acc

    if(acc.gt.eps) then
      ok = .true.
    else
      ! write(*,*) 'vector', i, 'converges', acc
      conv = conv + 1
      w(i) = t(i)
      acc_ev(i) = acc
      it_ev(i) = k
      eig_sum = eig_sum + w(i)
      i = i + 1
      if( eig_sum .ge. p_trace) ok = .true.
    end if
  end if
end do

n_ev = n_ev + conv
```

4 L'approche par blocs et la méthode de déflation : vers une solution plus efficace

Question 7

Le calcul de A^p est en o(n^3).
Le calcul de A^pV est en o(n^2m)+o(n^3), soit en o(n^3) également (car $m \leq n$).
Cependant, le calcul de AV est en o(n^2m).
Donc, il serait moins coûteux de faire une boucle (avec p itérations) réalisant $V = AV$ plutôt que de faire $V = A^pV$.

Question 8

Voir le code correspondant à la v2.
On remarque que la v2 est plus rapide que la v1 (environ 2 fois plus rapide dans les cas testés).

Question 9

L'algorithme v1 continue de faire converger des vecteurs ayant déjà convergé (au sens de la précision souhaitée), ainsi on observe une plus grande précision sur les valeurs propres les plus importantes (par rapport à la précision des valeurs propres obtenues à la dernière itération), car ce sont celles qui convergent le plus rapidement.

Question 10

La v2 ayant un fonctionnement identique à la v1 concernant le fait de ne pas freezer V_c, la même observation devrait pouvoir être faite.

Question 11

Voir le code correspondant à v3.

Question 12

On remarque que p affecte la rapiditié de l'algorithme et non pas suivant un comportement linéaire. En ffet, il semblerait qu'il existe une valeur de p otimale pour laquelle ce temps est minimum et que loin de cette valeur le temps d'exécution soit important.

Question 13

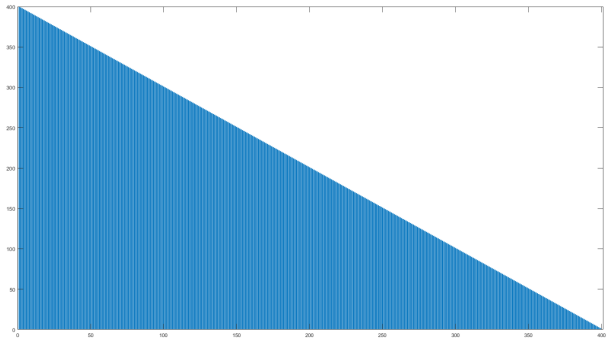

FIGURE 1 – Distribution des valeurs propres de la matrice 1

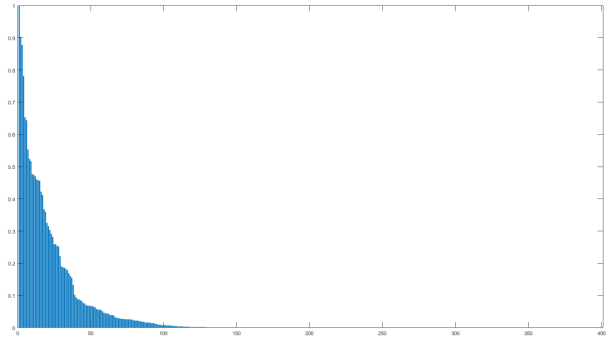

FIGURE 2 – Distribution des valeurs propres de la matrice 2

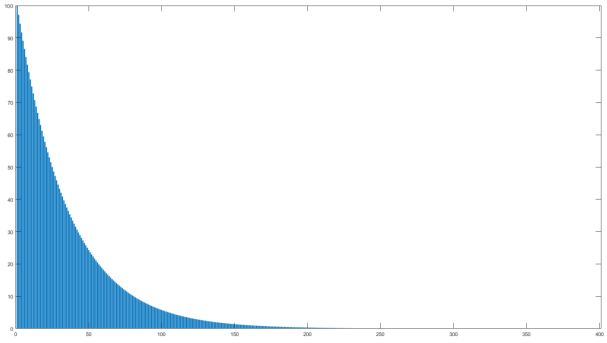

FIGURE 3 – Distribution des valeurs propres de la matrice 3

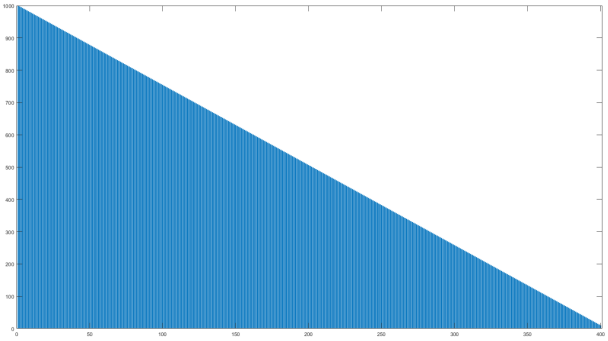

FIGURE 4 – Distribution des valeurs propres de la matrice 4

Pour la matrice 1 et la matrice 4, les valeurs de valeurs propres sont réparties de façon linéaire, mais la valeur maximale de la plus grande valeur propre de la matrice 4 est supérieure à la valeur maximale de la plus grande valeur propre de la matrice 1.

Pour la matrice 2 et la matrice 3, un nombre important de valeurs propres sont nulles et les valeurs de celles qui sont non nulles ne sont pas réparties de façon linéaire.

Question 14

En terme de temps d'exécution :
$v11 < v0 < v1 < v10 < v2 < v3$ pour $n = 200$, $m = 40$, $per = 0.5$, $imat = 2$
$v11 < v0 < v1 < v2 < v10 < v3$ pour $n = 400$, $m = 50$, $per = 0.5$, $imat = 3$
$v11 < v1 < v0 < v2 < v3 < v10$ pour $n = 300$, $m = 50$, $per = 0.5$, $p = 7$, $imat = 1$
$v11 < v0 < v1 < v10 < v2 < v3$ pour $n = 200$, $m = 40$, $per = 0.5$, $p = 7$, $imat = 2$
$v11 < v0 < v1 < v2 < v10 < v3$ pour $n = 400$, $m = 50$, $per = 0.5$, $p = 7$, $imat = 3$
$v11 < v1 < v0 < v2 < v3 < v10$ pour $n = 200$, $m = 40$, $per = 0.5$, $p = 7$, $imat = 4$ ...

On remarque que l'algorithme de la puissance itérée est le dernier en terme de temps d'exécution dans tous les cas.

Dans la majorité de cas, la version v3 est le premier en terme de temps d'exécution.

Dans certains cas, nous observons cependant que la version v10 s'avère être la meilleure.

5 Conclusion

Utilisant initialement la méthode de la puissance itérée avec déflation pour calculer les couples propres essentiels à une réduction de dimension, nous arrivons ici à la conclusion que le meilleur algorithme pour cela n'est pas ce dernier mais celui de la version 3.