Département Sciences du Numérique

TP- Projet 2 : Méthodes d'itération des sous-espaces

Samuel Boury – Clément Chanchevrier – Joël Ky

05 Avril 2019

Partie 1 : Limites de la méthode de la puissance itérée

Question 1 :

Quand on lance l'éxécution du programme principal, les résultats obtenus pour la puissance itérée avec deflation sont : Time = 0.65799999237060547
tandis que ceux pour DSYEV sont : Time = 0.10100000351667404 .
On remarque que l'algorithme DSEYV est meilleur.

Question 2 :

L'algorithme de la puissance itérée est plus lent à cause du calcul sequentiel des valeurs propres : en effet on a besoin de la 1ere valeur propre pour calculer la suivante et ainsi de suite alors que DSEYV calcule toutes les valeurs propres à la fois.

Partie 2 : Ameliorer la méthode de la puissance itérée pour calcul les vecteurs propres dominants

Question 3 :

En utilisant la méthode de la puissance itérée sur un ensemble de m vecteurs initiaux,on remarque que la matrice V résultante converge vers la norme de la matrice composée des m vecteurs initiaux, parce que l'algorithme s'arrête dès la 1ère itération (le résidu étant inférieur à la précision epsilon dès la 1ère itération.).

Question 4 :

Cela ne pose pas de problemes parce que H est une matrice de ayant un nombre de colonnes égal au nombre de valeurs propres que l'on veut retourner.

Question 5 :

Le script complété au niveau de la procedure **iter_v0** est le suivant :

```
104     y = v
105     call gram_schmidt(y, n, m, v)
106     !! ...
107
108     do while((acc .ge. eps) .and. (k .lt. maxit))
109
110       k = k + 1
111
112       !! Compute  y = a*v
113       call DGEMM('n', 'n', n, m, n, 1.D0, a, n, v, n, 0.D0, y, n)
114
115       !! Compute  h = v'*y
116       call DGEMM('t','n', m, m, n, 1.D0, v, n, y, n, 0.D0, h, m)
117
118       !! Compute the accuracy ||a*v - v*h||/||A|| == ||y - v*h||/||A||
119         !! Compute  aux = y-v*h
120       aux = y
121       call DGEMM('n', 'n', n, m, m, -1.D0, v, n, h, m, 1.D0, aux, n)
122       !! ...
123
124         !! Compute acc = ||aux|| / ||A||
125       acc = dlange('f', n, m, aux, n, work)/normF_A
126
127       write(*,'(" IT:",i5," -- Accuracy is: ",es10.2,a)',advance='no') k, acc, char(13)
128       !! V <- orthonormalization of y
129       call gram_schmidt(y, n, m, v)
130
131     end do
132
133     if(acc .lt. eps) then
134       ! compute the spectral decomposition of the Rayleigh quotient h
135       call DSYEV('V', 'U', m, h, m, w_aux, work, lwork, ierr)
136
136
137       if( ierr .ne.0 )then
138         write(*,'("Error in dsyev")')
139         ierr = -4
140         goto 999
141       end if
142
143       !! Sort in the decreasing order (dsyev returns eigenvalues in ascending order)
144       !! (we suppose that all the eigen values are positive)
145       do i=1, m
146         x(:, i) = h(:, m-i +1)
147         w(i) = w_aux(m-i +1)
148       end do
149
150       !! v = v*x
151       y = v
152       call DGEMM('n', 'n', n, m, m, 1.D0, y, n, x, m, 0.D0, v, n)
153       !! ...
154
```

Question 6 :

Orthonormalisation d'un ensemble de m vecteurs

```
124     !! Initial set of orthonormal vectors
125     call gram_schmidt(v, n, m, y)
```

Incrementation de k

```
130     k = k + 1
131
```

Y = A.V

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

Orthonormalisation des colonnes de Y

```
135     !! B. Orthonormalisation
136     call gram_schmidt(y, n, m, v)
137
```

Projection de Raleigh-Ritz

```
!! 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)
!!   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

!!   3. V = VX
y = v
call dgemm('n', 'n', n, m, m, done, y, n, x, m, dzero, v, n)
```

Etape d'analyse de la convergence

```
!! 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
!write(*,*) n_ev
```

Partie 3 : Vers un solveur efficace

Question 7 :

L'implémentation de l'accélération dans la procédure **iter_v2** est la suivante :

```
132
133       !! compute y = a^p
134       temp = a
135       do j = 1, p
136           call dgemm('n', 'n', n, n, n, done, temp, n, a, n, dzero, temp2, n)
137           temp = temp2
138       end do
139
140       do while((eig_sum .lt. p_trace) .and. (n_ev .lt. m) .and. (k .lt. maxit))
141
142         k = k + 1
143
144         !! compute  y = a^p*v
145         call dgemm('n', 'n', n, m, n, done, temp, n, v, n, dzero, y, n)
146
```

Question 8 :

L'éxécution du programme avec l'option **-disp 2** pour la procédure iter_v2 donne les résultats ci-dessous :

```
  5 First eigenvalues
 ========================
Eigenvalue    1:   100.000   accuracy : 0.730E-08   number of iterations :        17
Eigenvalue    2:    97.719   accuracy : 0.536E-08   number of iterations :        18
Eigenvalue    3:    95.490   accuracy : 0.604E-08   number of iterations :        20
Eigenvalue    4:    93.313   accuracy : 0.506E-08   number of iterations :        22
Eigenvalue    5:    91.184   accuracy : 0.187E-08   number of iterations :        22

 ||A*V_i - Lambda_i*V_i||/||A||
 ==============================
Eigenvalue    1: 0.699E-10
Eigenvalue    2: 0.132E-09
Eigenvalue    3: 0.105E-08
Eigenvalue    4: 0.506E-08
Eigenvalue    5: 0.187E-08

 ==================================
 Time =    7.6999999582767487E-002
 ==================================
```

En comparant ces résultats avec ceux donnés par l'éxécution de la même commande mais cette fois avec la procédure **iter_v1** on obtient les résultats ci dessous :

```
   5 First eigenvalues
  ========================
Eigenvalue    1:   100.000   accuracy : 0.730E-08   number of iterations :        34
Eigenvalue    2:    97.719   accuracy : 0.855E-08   number of iterations :        35
Eigenvalue    3:    95.490   accuracy : 0.936E-08   number of iterations :        39
Eigenvalue    4:    93.313   accuracy : 0.748E-08   number of iterations :        43
Eigenvalue    5:    91.184   accuracy : 0.278E-08   number of iterations :        43

 ||A*V_i - Lambda_i*V_i||/||A||
  ============================
Eigenvalue    1: 0.111E-09
Eigenvalue    2: 0.209E-09
Eigenvalue    3: 0.162E-08
Eigenvalue    4: 0.748E-08
Eigenvalue    5: 0.278E-08

 ================================
 Time =   0.10000000149011612
 ================================
```

Il ressort que la précision de la procédure **iter_v2** est bien plus importante que celle de **iter_v1** ce qui s'explique par l'accélération de la méthode **iter_v2** avec la multiplication par la matrice **a^p** ce qui a pour effet d'augmenter la valeur du rapport entre la plus grande valeur propre et la deuxième plus grande.

Question 9 :

La déflation implémentée dans l'algorithme aura pour effet de rendre la méthode beaucoup plus rapide étant donné que l'étape de la projection de Raleigh-Ritz sera réduite qu'aux colonnes qui n'ont pas encore convergées.

Question 10 : (non faite)

Partie 4 : Expérimentation numérique

Question 11 :

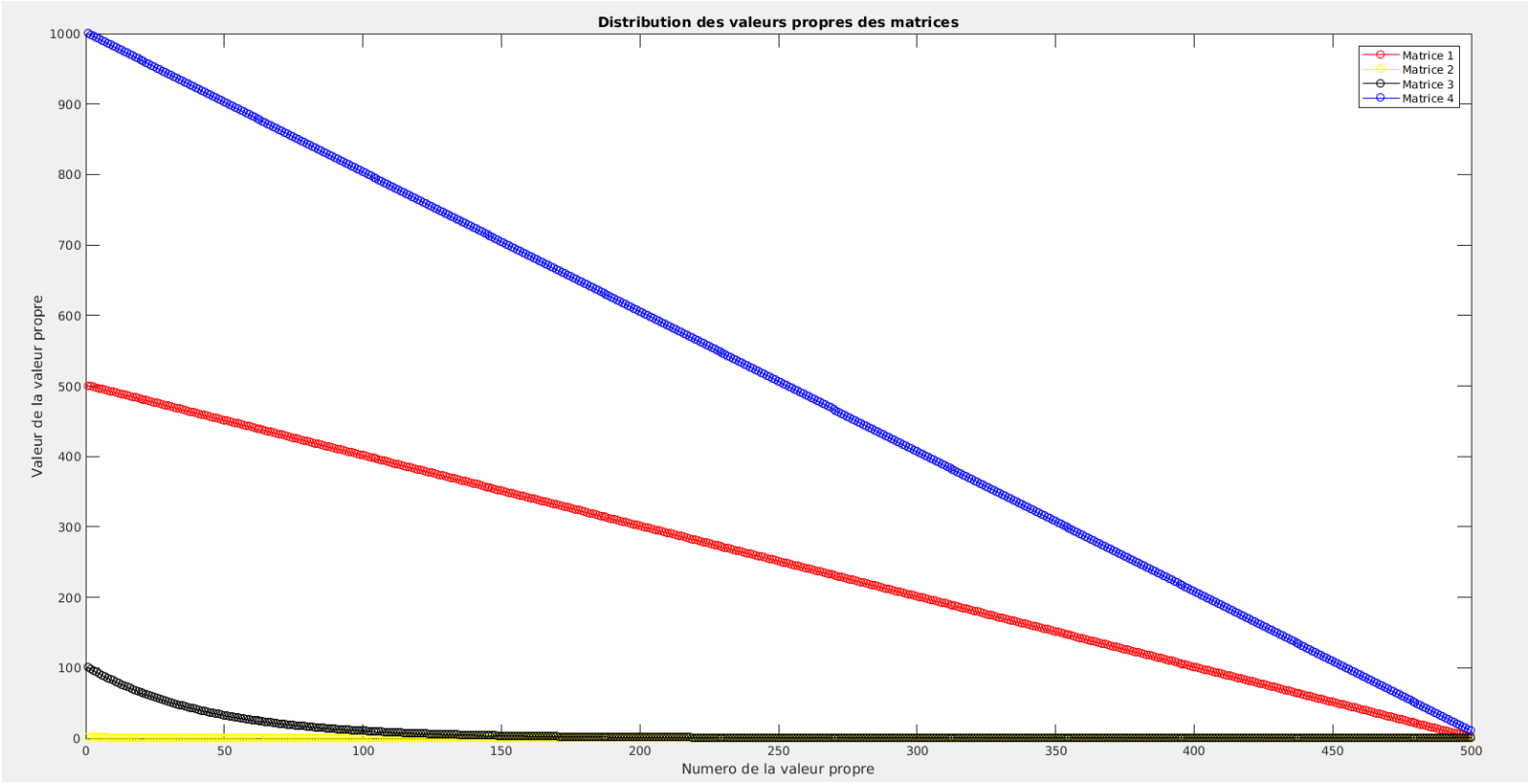
On peut observer que plus p augmente plus la précision des valeurs propres augmente, ce qui est normal car on augmente la puissance de a et donc on augmente la valeur du rapport entre la plus grande valeur propre et la deuxième plus grande valeur propre qui règle la

vitesse de convergence de la méthode, ce qui a donc pour effet d'augmenter la convergence de la matrice.

Question 12 :

L'observation de la distribution des valeurs propres des différentes matrices nous montre que la matrice de type 1 présente une distribution linéaire de valeurs propres. La matrice 4 aussi présente une distribution similaire.
Par contre les matrices de types 2 et 3 ont des valeurs propres qui ne sont pas linéairement distribuées. Une grande parties de leurs valeurs propres sont très petites.
Et ainsi les matrices comme celle du type 3 où le rapport entre la 1ère valeur propre et la 2nde sera plus important c'est-à-dire où les deux premières valeurs propres sont pas très proches, la méthode convergera plus rapidement. Par contre utiliser des matrices où les valeurs propres sont très proches l'une de l'autre, la méthode risque de ne pas faire ressortir les valeurs propres dans le bon ordre.

La figure suivante présente la distribution des valeurs propres de ces différentes matrices



Question 13 :

Pour une exécution avec en paramètres (/main -n 500 -m 20 -per 0.1) et une matrice de type 3, on remarque la méthode la plus performante en termes de temps d'exécution est la méthode **iter_v2,** qui s'exécute avec un temps d'execution de **7.9999998211860657E-002s** (alors que les temps d'exécution de DSEYV, iter_v0, iter_v1 sont respectivement de **0.14000000059604645s , 0.69199997186660767s, 0.10000000149011612s**).
Par contre la méthode la plus précise est bien DSEYV qui calcule les valeurs propres avec une précision de l'ordre de **10^(-15)**, tandis que les autres sont autour de **10^ (-8)** et **10^ (-9)**.
Il ressort que la méthode de la puissance itérée avec déflation (**iter_v0**) est la moins efficace aussi bien en termes de rapidité que de précision.