Rapport du Projet TP2 de Calcul Scientifique/Analyse de données : Calcul des Couples Propres

Théo PETIT
Thomas SADURNI
Thibault ROUX

Département Sciences du Numérique - Première année
2019-2020

Table des matières

Table des figures

1 Limites de la méthode de la puissance

Question 1 :

On remarque que la méthode de la puissance itérée n'est pas très efficace en temps de calcul, et ceux dans tous les cas que l'on a testé. Malgré le fait que la subroutine *dsyev* calcule toutes les valeurs propres, y compris celles qui sont inexploitables (car contenant trop peu de données), elle est plus efficace que l'algorithme de puissance itérée. Voici un tableau qui compare les temps de calculs des valeurs propres pour les deux méthodes et pour différentes matrices :

	Temps de calcul (s)	
Matrice(type et taille)/Valeurs propres calculées	Puissance itérée	Lapack DSYEV
type 1, taille 300, 20 vp	12.04	0.04
type 1, taille 100, 20 vp	0.57	0.003
type 2, taille 500, 10 vp	2.99	0.11
type 2, taille 500, 20 vp	7.95	0.11
type 2, taille 500, toutes vp	8.381	0.09
type 3, taille 300, 300 vp	12.956	0.029
type 4, taille 100, 100 vp	1.375	0.003

La subroutine *dsyev* est bien plus efficace, puisque son temps de calcul est nettement inférieur à celui de la puissance itérée avec déflation, alors que cette subroutine calcule toutes les valeurs propres de la matrice, contrairement à l'algorithme de la puissance itérée avec déflation.

Question 2 :

Le principal défaut de l'algorithme de la puissance itérée avec déflation est la mise à jour de la matrice A à chaque tour de boucle, qui est d'autant plus coûteuse que la taille de la matrice est grande. De plus, le temps de convergence (si convergence il y a) dépend du vecteur initial v. Enfin, la méthode de déflation a le défaut de calculer chaque vecteur propres de façon séparée, ce qui ralentit et cumule l'erreur si les valeurs propres ne sont pas précises.

2 Extension de la méthode de la puissance pour calculer les vecteurs propres dominants

Question 3 :

La matrice V converge vers une matrice dont chaque colonne est le vecteur propre associé à la plus grande valeur propre en module.

Question 4 :

Ce n'est pas un problème de calculer l'entière décomposition spectrale de H car c'est une matrice carrée de taille m, qui correspond au nombre de couples propres que l'on souhaite obtenir, et donc en général bien inférieur à n, taille de la matrice A.

Question 5 :

Voir $iter\_v0.f90$, les lignes modifiées sont les suivantes : 84, 85, 86, 109, 110, 117, 120, 124, 125, 128 et 132.

Question 6 :

Etape 1 - Boucle repeat :

repeat
...
until (PercentReached > PercentTrace or nev = m or k > MaxIter)

```
do while((eig_sum .lt. p_trace) .and. (n_ev .lt. m) .and. (k .lt. maxit))
```

Etape 2 - Incrémentation de k :

k = k + 1

```
k = k + 1
```

Etape 3 - Calcul de Y :

Compute Y such that Y = A·V

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

Etape 4 - Orthonormalisation :

V orthonormalisation of the columns of Y

```
call gram_schmidt(y, n, m, v)
```

Etape 5 - Projection de Rayleigh-Ritz :

Rayleigh-Ritz projection applied on matrix A and orthonormal vectors 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 6 - Analyse de la convergence :

Convergence analysis step : save eigenpairs that have converged and update PercentReached

```
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

call gram_schmidt(y, n, m, v)
```

3 subspace_iter_v2 et subspace_iter_v3 : vers un solveur efficace

Question 7 :

Le nombre d'opérations nécessaires au calcul de A^p est de l'ordre de $(p-1)n^3$ opérations. Puis n^3 opérations pour multiplier par V.
Pour le calcul de A^pV, Il est plus efficace de commencer par calculer AV puis d'effectuer des multiplications successives à gauche par A pour obtenir A^pV car cela revient à manipuler des matrices de dimensions $m*n, avecm <= n$ au lieu de matrices carrées de dimension n.

Question 8 :

Le code est visible dans le fichier $iter\_v2$. On calcul d'abord A^p pour ensuite multiplier par V.

Question 9 :

Dans la version $subspace\_iter\_v1$, de nombreuses itérations (plusieurs dizaines voire centaines) sont réalisées, ce qui entraîne des erreurs d'arrondis car les calculs sont faits sur des flottants.

Question 10 :

En revanche, dans $subspace\_iter\_v2$, le nombre d'itérations est divisé en moyenne par p, ce qui réduit fortement les erreurs de calculs car ils sont moins nombreux, et donc améliore la précision.

Question 11 :

Le code est visible dans le fichier $iter\_v3$.

4 Expériences numériques

Question 12 :

On remarque que plus le nombre p est important, plus le nombre d'itérations est faible, et donc plus le temps de calcul est faible, jusqu'à une certaine valeur de p pour laquelle le temps de calcul de A^p prévaut sur les itérations successives.

Question 13 :

La figure ci-dessous représente bien la différence spectrale des matrices de type $1, 2, 3$ et 4.

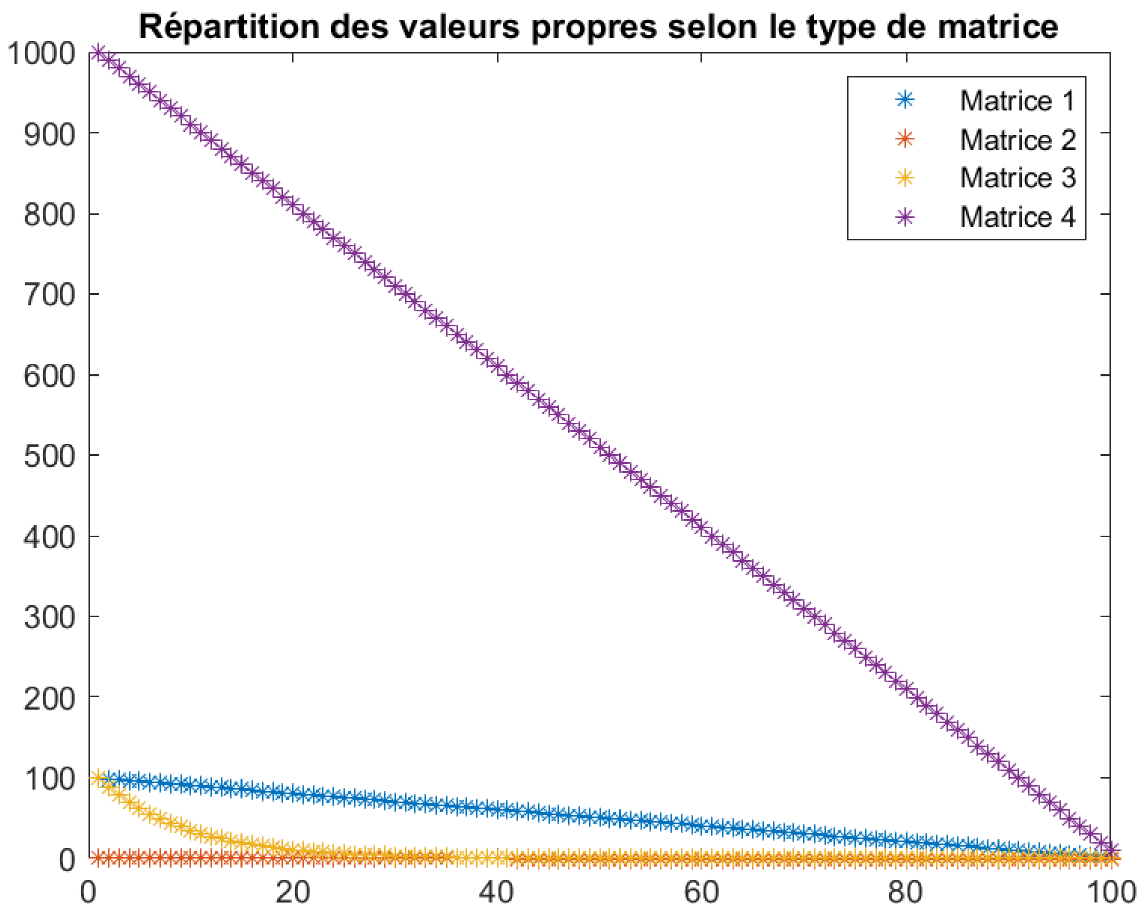

FIGURE 1 – Répartition spectrale des différentes matrices

Question 14 :

	Temps de calcul (s) (p=10)					
Matrice(type et taille)/Valeurs propres calculées	Puissance itérée	Lapack DSYEV	v0	v1	Block	Deflation
type 1, taille 300, 20 vp	12.04	0.04	3.93	5.05	0.71	2.79
type 1, taille 100, 20 vp	0.57	0.003	0.23	0.40	0.06	0.11
type 2, taille 500, 10 vp	2.99	0.11	0.32	0.57	0.23	0.29
type 2, taille 500, 20 vp	7.95	0.11	1.47	2.54	1.14	1.21
type 2, taille 500, toutes vp (31)	8.381	0.09	0.74	1.07	0.54	0.51
type 3, taille 300, 100 vp	1.45	0.029	3.78	0.11	*bcptrop*	0.56
type 4, taille 100, 50 vp	0.95	0.0002	0.63	0.28	0.10	0.0059

On remarque dans tous les cas que la méthode Lapack DSYEV fournie est la plus rapide, et la conjecture faite lors des premières questions est toujours vérifiée : la puissance itéré est peu efficace en temps. En ce qui concerne les méthodes implantées en Fortran, les méthodes par block et par déflation sont efficaces en temps. En revanche, la valeur de p influence le temps de calcul car plus p est important, plus le calcul de A^p est grand. Pour conclure, la méthode la plus efficace reste la méthode DSYEV, même si les deux méthodes implantées sont plus rapides pour des matrices de grande taille.