RAPPORT DE PROJET
DE
CALCUL NUMERIQUE

Tuteur : M.Boulier

JONGMANEE Tony
DEZE Florian

2015-2016
GIS3

SOMMAIRE

Introduction

Dans le cadre de notre formation en Génie Informatique et Statistique, en 3ème année, à Polytech Lille, nous avons suivi un cours de **Calcul Numérique** suivi d'un projet dont l'objectif étant d'appliquer les notions vues lors de ce cours.

Ce projet consiste à programmer en FORTRAN des méthodes permettant de calculer des valeurs et vecteurs propres de matrices, soit liées à l'analyse de composante principales, soient pour déterminer les avantages et inconvénients de ces méthodes. Les matrices traitées sont toutes réelles.

Dans ce projet, nous commencerons, dans une première partie, par étudier et comparer les méthodes de la matrice de covariance et de la décomposition singulière, liées à l'analyse de composante principales. Nous verrons ensuite, dans une seconde partie, l'algorithme QR et la méthode de la puissance inverse. Enfin nous recréerons la méthode de la décomposition singulière avec d'autres BLAS que celles utilisées dans la première partie.

1. Partie obligatoire

Dans cette partie, nous avons expliqué les tests permettant d'affirmer que nos programmes sur les méthodes de la matrice de covariance et de la décomposition en valeurs singulières fonctionnent. Nous avons vérifié nos résultats avec le logiciel Maple. Cette partie inclus une étude théorique liant les deux méthodes et une étude pratique comparant leur vitesse d'exécution et par conséquent leur différence de complexité.

Nous vérifions à chaque étape si les résultats obtenues par notre programme (obtenue avec la fonction : PRINT_MPL) sont les mêmes que ceux obtenus en Maple en les soustrayant l'un à l'autre. Il nous suffit ensuite d'observer que les résultats sont nuls.

1.1. Centrage et adimensionnement des données

Avant d'appliquer une méthode afin de déterminer les coordonnées des points, il est nécessaire dans un premier temps de centrer et d'adimensionner les données de la matrice A. Pour le centrage, il suffit de soustraire la moyenne d'une colonne à chaque élément de la colonne. Concernant l'adimensionnement des données, nous devons diviser les données d'une colonne par l'écart-type de celle-ci.

Ces deux points sont faciles à implémenter : nous utilisons des boucles POUR et en particulier la BLAS DDOT pour le calcul de l'écart-type.

Pour la vérification, nous comparons avec MAPLE la matrice obtenues après le centrage. Le code Maple est le suivant :

```
On suppose A déjà déclarée,
moy := <0,0,0,0,0,0,0,0,0,0,0,0>;
for j from 1 to 12 do
  moy[j]:=0;
for i from 1 to 17 do
  moy[j]:=A[i,j] + moy[j];
end do;
  moy[j] := moy[j]/17.;
end do;
moy;
for j from 1 to 12 do
   for i from 1 to 17 do
```

```
    A[i,j]:=A[i,j]-moy[j];
end do;
end do;
A;
```

Pour l'adimensionnnement, n'ayant pas réussi à obtenir un résultat en Maple, nous avons décidé de vérifier "à la main" sur une matrice plus petite 3x3.

1.2. Méthode de la matrice de covariance

Les tests ont été réalisés avec le fichier covariance.txt contenant toutes les lignes de code en Maple pour vérifier nos résultats.

Les étapes du programme à comparer avec Maple sont :

- La matrice de covariance
- Le calcul des valeurs propres
- Le calculs des vecteurs propres

Concernant les valeurs propres et vecteurs propres après factorisation de Schur, nous avons comparé les vecteurs propres correspondant au valeurs propres les plus élevées. Pour déterminer les vecteurs propres et valeurs propres en Maple nous utilisons la fonction Eigenvectors(COV) où COV est notre matrice de covariance.

Concernant la méthode utilisée pour obtenir nos résultats, elle suit les étapes données par le polycopié du sujet. Nous avons utilisé différentes BLAS afin de les obtenir. La matrice de covariance est déterminé avec la BLAS DGEMM qui réalise le produit entre A_TA. L'étape de la décomposition de Schur et l'étape servant à déterminer les deux vecteurs propres associés aux deux plus grandes valeurs propres se fait avec une seul BLAS DSYEV. Celle-ci nous donne les vecteurs propres rangés dans l'ordre croissant des valeurs propres. Ainsi, il suffit de récupérer les deux derniers vecteurs de la matrice COV, contenant les vecteurs propres après l'utilisation de DSYEV, avec les vecteurs VIN et VOUT. VIN contient le vecteur propre de la valleur propre maximale et VOUT le vecteur propre de la second plus grande valeur propre

1.3. Méthode par la décomposition en valeurs singulières

Pour cette méthode, nous avons vérifié avec Maple la matrice orthogonale VT contenant les vecteurs singuliers obtenue après l'utilisation de la BLAS DGESVD, puis la matrice VA contenant la transposé de VT et enfin les vecteurs V1 et V2 qui sont résultats finaux. Cependant, une fois que nous ayons multiplié la matrice A par les vecteurs V1 et V2, nous avons obtenu des mauvais résultats. Après plusieurs vérifications, A était modifié par DGESVD. C'est pour cela que nous avons créé la matrice B.

Les résultats attendus sont différents des résultats obtenues. il se peut que nous obtenions des résultats opposés (valeurs négatives au lieu de positives et inversement). Pour confirmer l'exactitude de nos résultats, nous prenons en compte la valeur absolue de chaque élément de notre résultante. Ainsi, notre programme rend le résultat attendu.

On peut également vérifier les valeurs singulières et les vecteurs singuliers via la fonction SingularValues(A, output['U','S','Vt']).

Grâce à la BLAS DSYEV, le programme fut plutôt facile à écrire. Il nous permet de remplacer les étape de calculs de la covariance et de la factorisation de Shur en une seule BLAS. Un code plus court ne signifie pas qu'il est plus rapide, ni qu'il est plus précis dans ces résultats.

1.4. Comparaison des vitesses des deux méthodes

Nous avons inséré dans nos programmes la fonction CPU_TIME permettant de calculer la vitesse d'exécution d'un programme FORTRAN (fonction inspiré du TD1). Les résultats obtenues varient d'une machine à l'autre si celles-ci ont des vitesses d'exécution différentes. Les résultats suivant ont été obtenus avec nos ordinateurs personnelles (les ordinateurs de l'école ne donnant pas de résultats significatifs).

Nous obtenons :

Pour la matrice covariance : 4.00000811E-03

Par la décomposition en valeurs singulières : 8.00096989E-03

Nous remarquons que, pour une matrice de taille 12x17, le temps d'exécution de la décomposition en valeurs singulière est deux fois plus long qu'avec la méthode de la matrice de covariance. Nous avons réalisé les mêmes tests avec une matrice plus petite de taille 10x15 pour vérifier que si les temps les temps d'exécutions sont proportionnelles. Avec une matrice plus petite nous obtenons :

Pour la matrice covariance : 4.00000811E-03

Par la décomposition en valeurs singulières : 4.00000811E-03

Les résultats sont similaires, donc nous pouvons conclure qu'ils ne sont pas reliés par un facteur 2. Par contre, nous pouvons observer qu'il y a une réelle différence entre le premier et le second résultats pour la décomposition en valeurs singulières, alors qu'avec la matrice covariance il n'y a pas de différence. La complexité de la décomposition singulière est donc plus forte que celle de la matrice de covariance.

1.5. Analyse Théorique : Explication des résultats similaires

La matrice de covariance C est égale à $\alpha A_T A$ avec $\alpha=1/(n-1)$ et n le nombre de colonne de A. Nous pouvons en déduire que $A_T A=(U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^T U^T\ U\Sigma V^T = V\Sigma^T\Sigma V^T$. Nous en concluons que A^TA et $\Sigma^T\Sigma$ ont des valeurs propres similaires. De plus Σ contient les valeurs singulières σ_i et elle est diagonale. Ainsi, $\Sigma^T\Sigma$ donne les valeurs singulières σ_i aux carrés qui correspondent aux valeurs propres de A.

A partir de la racine des valeurs propres de la matrice de covariance, nous retrouvons les valeurs singulières de la SVD, et inversement en élevant les valeurs singulières au carrées.

Pour conclure, la méthode de la matrice de covariance possède des contraintes, la matrice C doit être carrée et diagonalisable. Dans notre démarche $C=A^TA$ nous donne une matrice carrée, symétrique, donc la matrice est forcément diagonalisable. Cependant, cela implique un traitement supplémentaire sur la matrice A. En revanche, la décomposition en valeurs singulières peut s'effectuer sur n'importe quel matrice A de dimension mxn. Elle a donc moins d'inconvénients concernant les propriétés de la matrice A. Si nous comparons les coûts des algorithmes (grâce aux temps d'exécutions), nous avons vu que la méthode de la matrice de covariance est moins coûteuse, malgré ce traitement supplémentaire A^TA.

2. Partie facultative

Dans cette partie, nous verrons l'algorithme QR permettant de calculer les valeurs propres d'une matrice carrée symétrique et la méthode de la puissance inverse qui calcule les vecteurs propres d'une matrice à partir de ses valeurs propres. Nous expliquerons en détail notre version de la décomposition en valeurs singulières. Les tests sur les deux premiers programmes ont été effectués sur une même matrice A ci-dessous et tous les résultats ont été vérifiés sur Maple. Concernant la dernière partie, la matrice utilisé est la me que celle présenté sur le sujet de projet

$$A := \begin{bmatrix} 3. & 4. & 5. & 7. \\ 4. & 89. & 8. & 15. \\ 5. & 8. & 67. & 34. \\ 7. & 15. & 34. & 12. \end{bmatrix}$$

La matrice A est carrée et symétrique.

2.1. Algorithme QR

L'agorithme QR permet de déterminer les valeurs propres d'une matrice.
Nous avons repris l'algorithme SHIFT_QR se trouvant dans le polycopié du cours, cependant, il reste quelques lignes à implémenter.
De plus, pour appliquer cet algorithme, nous transformons la matrice A en une matrice tridiagonale : utilisation de Hessenberg avec la BLAS DSYTRD.

Nous avons créé un sous-programme appelé SHIFT qui permet de soustraire ou additionner la diagonale d'une matrice avec un réel en fonction de l'option choisie. Nous l'utilisons ici sur la matrice A afin de soustraire à sa diagonale le réel MU aux cases d'indice K0 à K1. Puis, nous stockons dans une matrice ce bloc de A. Il suffit ensuite d'utiliser les BLAS DGEQRF (factorisation QR) et DORMQR (calcul du produit R.Q). Et enfin, nous réutilisons une nouvelle fois le sous-programme SHIFT au bloc qui devra par la suite être inséré dans A.

Pour vérifier nos résultats, nous avons créé dans algoQR.txt un code Maple pour obtenir les valeurs propres de la matrice A. Après l'exécution de l'algorithme QR, A devient une matrice diagonale et ses valeurs propres sont sur sa diagonale. Il suffit donc de comparer ces valeurs propres

avec ceux de obtenues avec Maple.
Les valeurs propres de A que l'ont doit trouver sont :

[-6.92823670367070576 + 0. I]
[ ]
[3.78829111330153401 + 0. I]
[ ]
[101.357870847564783 + 0. I]
[ ]
[72.7820747428043830 + 0. I]

Ces résultats sont trouvés grâce à la fonction Eigenvalues(A). Les 0.I ne sont pas à prendre en compte dans le résultat (cela vaut 0).

2.2. Méthode de la Puissance Inverse

Pour cette méthode, nous avons l'algorithme suivant :

M=(A-μI)
V0= vecteur quelconque normalisé
POUR I DE 1 A N FAIRE
$V_i=M^{-1}.V_{i-1}$
$V_i=V_i/||V_i||_2$
FIN POUR

A est notre matrice de départ, V un vecteur quelconque convergeant vers un vecteur propre de A associé à la valeur propre approximative μ de A.

Cet algorithme pose problème pour le calcul d'inverse de matrice. Cette transformation peut s'avérer long à implémenter et serai très coûteux lorsque la matrice est de grande taille. Par un calcul très simple, nous éliminons l'inversion : $V_i=M^{-1}.V_{i-1} \Leftrightarrow M.V_i=M.M^{-1}.V_{i-1} \Leftrightarrow M.V_i=I.V_{i-1} \Leftrightarrow M.V_i=V_{i-1}$

Pour résoudre cette équation, nous la transformant en un système triangulaire. Pour cela nous utilisons la factorisation LU sur M : M=L.U. Nous obtenons donc l'équation : $L.U.V_i=V_{i-1}$ qui peut se résoudre en deux systèmes triangulaires.
On pose $U.V_i=y$. Cela nous donne l'équation suivante $L.y=V_{i-1}$. Il suffit ensuite de résoudre cette équation. Nous trouvons donc les valeurs de y. Enfin nous résolvons
$U.V_i=y$.

Nous obtenons ainsi l'algorithme suivant :

M=(A-μI)
M=L.U
V0= vecteur quelconque
POUR I DE 1 A N FAIRE
$L.y=V_{i-1}$
$U.V_i=y$
$V_i=V_i/||V_i||_2$
FIN POUR

En fortran, nous avons utilisé un entier ISEED et la fonction RAND_M pour générer un vecteur v aléatoire. Pour déterminer μ, nous avons utilisé la fonction DSYEV afin de connaître les valeurs propres de A et en les plaçant dans un vecteur MU. Ces vecteurs propres sont les mêmes que ceux donnés par la l'algorithme QR. Il est également possible d'utiliser l'algorithme QR pour déterminer les valeurs propres de A, cependant par manque de temps cela n'a pu être réalisé. Notre programme principale applique dans une boucle allant de 1 à M (taille de la matrice) l'algorithme de la puissance inverse avec notre sous-programme PUISSANCEINV et une valeurs MU différentes à chaque fois. Il affiche à chaque tour de boucles le vecteur propre V.

Dans notre sous-programme PUISSANCEINV, nous appliquons le second algorithme décrit précédemment. Le décalage par la valeur propre de A se fait avec le sous programme SHIFT utilisé pour l'agorithme QR. Concernant la factorisation L.U, nous utilisons la BLAS DGETRF. Cependant les deux résolutions de systèmes triangulaires se font en une seule fois avec DGETRS.

Pour tester ce programme, nous avons crée un code en Maple dans puissanceinv.txt calculant les valeurs propres de la matrice A.
Nous avons utilisé la fonction Eigenvectors(A) en Maple pour obtenir ce résultat :

V1	V2
[-0.384244871856672387 + 0. I]	[0.918506576834581701 + 0. I]
[-0.0870017395044811875 + 0. I]	[-0.0763550328807058171 + 0. I]
[-0.354584508840359913 + 0. I]	[-0.0763550328807058171 + 0. I]
[0.847971934597769761 + 0. I]	[0.311885105485740943 + 0. I]

V3	V4
[0.0823077455807778441 + 0. I]	[0.0438974006164139646 + 0. I]
[0.776357489387504928 + 0. I]	[-0.619572154755934300 + 0. I]
[0.526237239952835334 + 0. I]	[0.737631978113550191 + 0. I]
[0.526237239952835334 + 0. I]	[0.264768631326487591 + 0. I]

2.3. Méthode par la décomposition en valeurs singulières détaillée

Contrairement à la méthode utilisé dans la partie obligatoire, nous n'allons pas utiliser la BLAS DGESVD. En effet, nous allons reproduire nous même la décomposition en valeurs singulières. On commence d'abord par copié la matrice A dans une matrice B afin de ne pas effacer la matrice A de départ. Cette matrice est la même que celle présenté dans notre sujet de projet :

A :=
<<-.7000000000 |-4.360000000 |-.2000000000E-01 >,
<0.7000000000 |-1.240000000 |-4.180000000 >,
< 2.300000000 |-1.960000000 |-3.220000000 >,
<-2.300000000 |-3.640000000 |-.9800000000 >>;

Nous souhaitons transformer, la matrice B en matrice bidiagonale supérieure. Pour cela on applique la BLAS DGBBRD. Cette BLAS requière un traitement préliminaire sur les éléments de la matrice B que nous réalisons sur une matrice AB. Cette matrice AB est tridiagonale.

$$AB = \begin{bmatrix} X & X & 0 & 0 & 0 \\ X & X & X & 0 & 0 \\ 0 & X & X & X & 0 \\ 0 & 0 & X & X & X \\ 0 & 0 & 0 & X & X \end{bmatrix}$$

ou

$$AB = \begin{bmatrix} 0 & 0 & 0 & X & X \\ 0 & 0 & X & X & X \\ 0 & X & X & X & 0 \\ X & X & X & 0 & 0 \\ X & X & 0 & 0 & 0 \end{bmatrix}$$

Comme nous n'arrivions pas à vérifier cette matrice ne Maple, nous nous sommes contenté de regarder si elle était tridiagonale. Le résultat de notre matrice AB doit être correcte si sa transformation en matrice bidiagonale possède les bonnes valeurs.

Après utilisation de DGBBRD, la matrice AB est bidiagonale. Le résultats attendu est :

B :=
<< 3.400000000 | 2.964705882 | 0.000000000 >,
< 0.000000000 | 2.941176471 |0.2888314588E-15 >,
< 0.000000000 | 0.000000000 | 7.000000000 >>;

Ce résultat s'obtient grâce à deux vecteurs D et E, où D représente la diagonale de B et E ses diagonales supérieure et inférieure. Ces vecteurs sont insérés dans AB.

A présent, nous insérons les valeurs de AB dans une matrice T deux fois plus grande comme indiqué sur le polycopié.

```
T :=
<< 0.000000000 | 3.400000000 | 0.000000000 | 0.000000000 | 0.000000000 | 0.000000000 >,
 < 3.400000000 | 0.000000000 | 2.964705882 | 0.000000000 | 0.000000000 | 0.000000000 >,
 < 0.000000000 | 2.964705882 | 0.000000000 | 2.941176471 | 0.000000000 | 0.000000000 >,
 < 0.000000000 | 0.000000000 | 2.941176471 | 0.000000000 | 0.000000000 | 0.000000000 >,
 < 0.000000000 | 0.000000000 | 0.000000000 | 0.000000000 | 0.000000000 | 7.000000000 >,
 < 0.000000000 | 0.000000000 | 0.000000000 | 0.000000000 | 7.000000000 | 0.000000000 >>;
```

Ce résultat n'a pas était calculé en Maple car il suffit juste de regarder les placement des valeurs. Il n'y a pas de calcul, donc il est assez rapide de constater si le résultat est incorrect.

Cette matrice est tridiagonale symétrique, avec une diagonale remplie de 0.
On détermine les valeurs propres de T, avec la BLAS DSYEV. En Maple, les valeurs sont calculés à partir de la fonction Eigenvalues(T) :

```
[     -5. + 0. I      ]
[                     ]
[     -2. + 0. I      ]
[                     ]
[     5. + 0. I       ]
[                     ]
[     2. + 0. I       ]
[                     ]
[     7. + 0. I       ]
[                     ]
[     -7. + 0. I      ]
```

On remarque que l'on a les mêmes valeurs propres en double (à un signe près)

Ces valeurs propres sont égales aux valeurs singulières de AB qui sont semblable à celles de A.

On détermine les vecteurs propres en réarrangent l'ordre des éléments de T de telle manière à obtenir T=√2.P.T avec P la matrice décrite sur le sujet de projet, et en les plaçant dans les matrice U et V.

T=√2.P.T :=

<<-0.000000000 |0.6000000000 |0.8000000000 |-.8000000000 |0.6000000000 | 0.000000000 >,
<-0.000000000 |0.8000000000 |-.6000000000 |0.6000000000 |0.8000000000 | 0.000000000 >,
<-1.000000000 |-.0000000000 |0.0000000000 |0.0000000000 |0.0000000000 | 1.000000000 >,
<-0.000000000 |-.8823529412 |-.4705882353 |-.4705882353 |0.8823529412 | 0.000000000 >,
<0.0000000000 |-.4705882353 |0.8823529412 |0.8823529412 |0.4705882353 |0.0000000000>,
< 1.000000000 |0.0000000000 |-.0000000000 |-.0000000000 |-.0000000000 | 1.000000000 >>;

Comme la multiplication par la matrice P ne fait que réaranger l'ordre de la matrice, nous n'avons pas utilisé Maple pour vérifier le résultat. Une fois que les résultat attendue était correcte, il ne restait plus qu'à multiplier par √2.

Ce même raisonnement à été appliqué sur U et .

U :=

<<-.4705882353 |0.8823529412 | 0.000000000 >,
<0.8823529412 |0.4705882353 |0.0000000000 >,
<-.0000000000 |-.0000000000 | 1.000000000 >>;

V :=

<<-.8000000000 |0.6000000000 | 0.000000000 >,
<0.6000000000 |0.8000000000 | 0.000000000 >,
<0.0000000000 |0.0000000000 | 1.000000000 >>;

On calcule ensuite la transposé de V que l'on nomme VT. Ici la matrice V est symétrique donc V=VT.

On vérifie que celle-ci sont bien orthogonale avec Maple en appliquant : Transpose(V).V et Transpose(U).U. On obtient une matrice d'identité pour chacune de ces matrices. Elles sont donc orthogonales.

Enfin pour obtenir les vecteurs singuliers de A, on applique la formule :

$A = Q_L.U.B.VT.Q_R$ où Q_L=Q et Q_R=PT dans notre programme. Ces matrices proviennent de la BLAS DGBBRD utilisé au début de notre programme.

$Y = VT.Q_R :=$

```
<<-.8000000000   |0.3600000000   |-.4800000000   >,
 <0.6000000000   |0.4800000000   |-.6400000000   >,
 < 0.000000000   |0.8000000000   |0.6000000000   >>;
```

$Z= Q_L.U : =$

```
<<-.5000000000   |-.5000000000   |-.5000000000   >,
 <0.5000000000   |0.5000000000   |-.5000000000   >,
 <-.5000000000   |0.5000000000   |-.5000000000   >,
 <0.5000000000   |-.5000000000   |-.5000000000   >>;
```

Ces résultats sont similaires à ceux trouvés dans le polycopié.

Pour prouver que les matrices Z et Y sont bien orthogonales, on les multiplie par leurs transposés

Transpose(Z).Z :=

```
[1.   0.   0.]
[             ]
[0.   1.   0.]
[             ]
[0.   0.   1.]
```

La matrice obtenue est une matrice d'identité prouvant bien que Z est orthogonale.

Un calcul similaire à été réalisé sur Y et le résultat fût le même.

Pour vérifier nos résultat avec Maple, nous copions ces matrices et nous réalisons le calcul suivant :

A := Z.S.Y où S est une matrice diagonale composées des valeurs propres de B (2, 5 et 7).

Le code est le suivant:

```
Sigma := <2,5,7>;
S := DiagonalMatrix(Sigma);
```

A la fin, on obtient :

A := Z.S.Y;

```
                [-0.699999999999999956   -4.36000000000000032   -0.0199999999999999068]
                [                                                                      ]
                [0.699999999999999956    -1.24000000000000022   -4.17999999999999971   ]
A :=            [                                                                      ]
                [ 2.29999999999999984    -1.96000000000000019   -3.21999999999999975   ]
                [                                                                      ]
                [-2.29999999999999984    -3.64000000000000013   -0.979999999999999872  ]
```

Le résultat obtenu est celui de notre matrice A de départ.

Conclusion

Tous les programmes demandés ont été réalisés. Les résultats obtenus sont bien ceux attendus.

Concernant la partie obligatoire, nous concluons que la méthode de la matrice de covariance est plus efficace que la méthode de décomposition en valeurs singulières en terme de coût, même si elle est plus contraignante.

Pour la partie facultative, nous n'avons pas eu le temps d'utiliser l'algorithme QR dans notre programme de puissance inverse.