Rapport n°2 du projet d'analyse de données et de calcul scientifique

Cédric Legoupil, Flavie Misarsky et François Testu

Avril 2022

Sommaire

1 Introduction

2 Les limites de la méthode de la puissance itérée

2.1 Question 1

Après avoir exécuté le fichier test_v11.m, nous obtenons les résultats suivant :

Temps eig = 1.000e-2 secondes

Temps puissance iteree = 5.360 secondes

On constate que le temps d'exécution avec la puissance itérée est bien plus grand que celui avec la fonction eig. La méthode de la puissance itérée n'est donc pas la plus efficace en temps de calcul.

2.2 Question 2

Dans cette question nous avons changé l'algorithme, nous l'avons réarrangé dans le but de diminuer son temps d'éxecution. Nous avons donc implanter l'algorithme suivant :

Algorithme de la puissance itérée réarrangé :

v = randn(n,1)
nombre_iteration = 1
$\beta = v' * z$
$z = A * v$
$norme = \|\beta * v - z\|/\|\beta\|$
Tant que ($norme > eps$ **et** $nombre\_iteration < maxit$)
    y = A * v
    $\beta = v' * y$
    $norme = \|\beta * v - y\|/\|\beta\|$
    $v = y/\|y\|$
    nombre_iteration = nombre_iteration + 1
Fin Tant Que

Boucle de l'algorithme de la puissance itérée réarrangé :

```
while(norme > eps && nb_it < maxit)
  y = A*v;
  beta = v' * y;
  norme = norm((beta*v - y), 2)/norm(beta,2);
  v = y/norm(y,2);
  nb_it = nb_it + 1;
end
```

Figure 1: Nouvel algorithme de la puissance itérée

Après le réarrangement nous obtenons le temps d'éxecution suivant :

```
******* calcul avec la méthode de la puissance itérée ******

Temps puissance itérée = 3.080e+00
```

Figure 2: Résultats obtenus après le réarrangement de l'algorithme

2.3 Question 3

Le principal inconvénient de la méthode de déflation est que le critère d'arrêt (pourcentage d'informations récupérées) peut être très dur à atteindre puisque les valeurs propres exclues à chaque fois sont les plus importantes. Il ne reste que les moins importantes, qui font progresser le pourcentage très lentement.

3 Extension de la méthode de la puissance pour calculer les vecteurs dominants de l'espace propre

3.1 Question 4

Si on applique la méthode de la puissance à un ensemble de m vecteurs, V convergera vers une base orthonormale de vecteurs propres.

3.2 Question 5

A $\in \mathrm{R}^{n*n}$, V $\in \mathrm{R}^{n*m}$. Or, $H = V^T * A * V$. Donc, H $\in \mathrm{R}^{m*m}$.

On cherche à calculer m couples propres de A. Par le calcul de tous les couples propres de H, on calcule ainsi m couples propres de A car une valeur propre de

H est une valeur propre de A et on retrouve les vecteurs propres associés grâce à la formule suivante $V_{out} = V * X$. (La démonstration de la formule est donnée dans le poly)

V étant une base orthonormée de dimension m et X étant la concaténation des vecteurs propres de H.

3.3 Question 6

```
% rappel : conv = invariance du sous-espace V ||AV - VH||/||A|| <= eps
while (~conv & k < maxit),

    k = k + 1;

    % calcul de Y = A.V
    Y = A * V;

    % calcul de H, le quotient de Rayleigh H = V^T.A.V
    H = V' * Y;

    % vérification de la convergence
    conv = (norm(A*V - V*H, 'fro') / norm(A ,'fro') <= eps);

    % orthonormalisation
    V = mgs(Y);

end
```

Figure 3: Adaptation de subspace_iter_V0 pour faire apparaître l'algo 2

3.4 Question 7

```
52      while (~conv & k < maxit),
53
54          k = k+1;
55          %% Y <- A*V
56          Y = A*Vr;
57          %% orthogonalisation
58          Vr = mgs(Y);
59
60          %% Projection de Rayleigh-Ritz
61          [Wr, Vr] = rayleigh_ritz_projection(A, Vr);
62
63          %% Quels vecteurs ont convergé à cette itération
64          analyse_cvg_finie = 0;
65          % nombre de vecteurs ayant convergé à cette itération
66          nbc_k = 0;
67          % nb_c est le dernier vecteur à avoir convergé à l'itération précédente
68          i = nb_c + 1;
```

Figure 4: Première partie de l'implantation de l'algorithme 4 (l'analyse de la convergence n'est qu'initialisée)

```
        while(~analyse_cvg_finie),
            % tous les vecteurs de notre sous-espace ont convergé
            % on a fini (sans avoir obtenu le pourcentage)
            if(i > m)
                analyse_cvg_finie = 1;
            else
                % est-ce que le vecteur i a convergé

                % calcul de la norme du résidu
                aux = A*Vr(:,i) - Wr(i)*Vr(:,i);
                res = sqrt(aux'*aux);

                if(res >= eps*normA)
                    % le vecteur i n'a pas convergé,
                    % on sait que les vecteurs suivants n'auront pas convergé non plus
                    % => itération finie
                    analyse_cvg_finie = 1;
                else
                    % le_vecteur i a convergé
                    % un de plus
                    nbc_k = nbc_k + 1;
                    % on le stocke ainsi que sa valeur propre
                    W(i) = Wr(i);

                    itv(i) = k;

                    % on met à jour la somme des valeurs propres
                    eigsum = eigsum + W(i);

                    % si cette valeur propre permet d'atteindre le pourcentage
                    % on a fini
                    if(eigsum >= vtrace)
                        analyse_cvg_finie = 1;
                    else
                        % on passe au vecteur suivant
                        i = i + 1;
                    end
                end
            end
```

Figure 5: Deuxième partie de l'implantation de l'algorithme 4 (l'analyse de la convergence est effectuée ici)

4 Subspace_iter_v2 et subspace_iter_v3 : vers un solveur efficace

4.1 Question 8

Tout d'abord calculons A * A :

Pour cela, calculons chacun des coefficients dans des boucles for :

for i entre 1 et n répéter
  for j entre 1 et n répéter
    for k entre 1 et n répéter
      $c_{ij} = a_{ij} * b_{kj}$
    end
  end
end

La complexité du produit A*A est donc de n^3, ainsi la complexité de A^3 est de $2n^3$. Au final, la complexité de A^p est donc de (p-1)n^3.

En ce qui concerne la complexité du produit A^p*V, nous procédons de la même manière dans notre raisonnement. La matrice V étant de taille n*m la complexité d'un produit d'une matrice carrée de taille n*n par V correspond à :

for i entre 1 et m répéter
  for j entre 1 et n répéter
    for k entre 1 et n répéter
      $c_{ij} = a_{ij} * b_{kj}$
    end
  end
end

La complexité du produit d'une matrice carrée de taille n*n par V est donc m*n^2.

Et ainsi le produit de A^p de taille n*n par V à une complexité de

$$(p-1)n^3 + m * n^2$$

Pour reduire le coût de calcul on pourrait enregistrer la valeur de A^2, voir de A^3, et les réutiliser pour calculer A^p au lieu de faire (p-1) produit de A.

4.2 Question 9

On a modifié dans le programme subspace_iter_v2, A.V par $A^p.V$ comme demandé.

4.3 Question 10

```
******* calcul avec subspace iteration v2 ******

Temps subspace iteration v2 = 2.600e-01
Nombre d'itérations : 7
```

Figure 6: test de l'algorithme subspace_iter_v2 modifié avec p=5)

```
******* calcul avec subspace iteration v2 ******

Temps subspace iteration v2 = 1.800e-01
Nombre d'itérations : 5
```

Figure 7: test de l'algorithme subspace_iter_v2 modifié avec p=8)

On a réduit d'un facteur 10 le temps de calcul grâce à cette méthode.

4.4 Question 11

Pour la méthode subspace_iter_v1 il peut arriver que le nombre d'itérations ne soit pas suffisant pour atteindre la précision eps demandée.

4.5 Question 12

On fera moins de calcul au niveau des produits donc la complexité sera plus faible. Cela implique que le temps d'exécution et la convergence seront plus rapides.

4.6 Question 13

```
******* calcul avec subspace iteration v2 ******

Temps subspace iteration v2 = 3.500e-01
Nombre d'itérations : 7
```

Figure 8: Test de l'algorithme subspace_iter_v2 avec p=7)

```
******* calcul avec subspace iteration v3 ******

Temps subspace iteration v3 = 2.000e-01
Nombre d'itérations : 7
```

Figure 9: Test de l'algorithme subspace_iter_v3 modifié avec p=7)

On observe que l'on a quasiment divisé par 2 le temps d'exécution du programme ce qui est cohérent avec la Q12.

5 Numerical experiments

5.1 Question 14

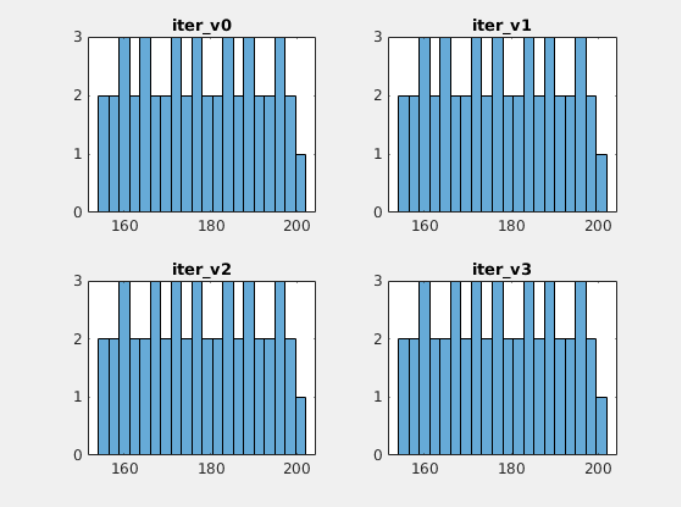

Figure 10: Distribution des valeurs propres selon les différents algorithmes)

On a tracé des histogrammes présentant la distribution des valeurs propres selon les différents algorithmes. Ces histogrammes présentent de grandes similitudes malgré des incertitudes de calcul liées aux différentes méthodes et approximations de l'ordinateur. Ces résultats semblent montrer que les 4 méthodes sont valables pour calculer les valeurs propres de plus fort poids.

5.2 Question 15

L'algorithme le plus rapide est celui de iter_v3 car il utilise à la fois la modification avec la puissance qui permet d'améliorer la rapidité du calcul mais aussi ne recalcule pas les produits de $A * V_c$ invariants.
Les algorithmes iter_v1 et iter_v2 sont des algorithmes moins optimisés car ils présentent de la redondance et des calculs de produits lourds et inutiles.
Pour ce qui est de iter_v0, il calcule obligatoirement tous les vecteurs et valeurs propres de la matrice ce qui n'est pas forcément nécessaire selon ce que l'on veut faire de ces valeurs.