Séance-Projet 2

Subspace Iteration Methods

Nihal Belmakhfi, Augustin Boulard, Héloïse Lafargue

Calcul Scientifique et Analyse de Données
Département Sciences du Numérique - Première année
2021-2021

Table des matières

Table des figures

1 Rapport

Question 1 : Nous obtenons par exemple les temps d'éxécution suivants :

```
******* calcul avec eig ******

Temps eig = 2.000e-02
Qualité des valeurs propres (par rapport au spectre de la matric
Qualité des couples propres = [5.819e-16 , 1.106e-13]

Matrice 200 x 200 - type 1

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

Temps puissance itérée = 4.940e+00
```

FIGURE 1 – Comparaison temps d'éxécution

eig est plus rapide que la puissance itérée pour des raisons algorithmiques (sur Matlab eig a été optimisé, codage en C).

Question 2 :

```
56          while(norme > eps && nb_it < maxit)
57            v = z / norm(z,2);
58            z = A*v;
59            beta = v'*z;
60            norme = norm(beta*v - z, 2)/norm(beta,2);
61            nb_it = nb_it + 1;
62          end
```

FIGURE 2 – Réarrangement de l'algorithme

Question 3 : Le principal inconvénient de la méthode est qu'elle demande, en plus de calculer un produit matriciel à chaque itération, de normaliser le vecteur z, ce qui augmente le temps de calcul.

Question 4 : Avec cette méthode, la matrice V converge vers une matrice de m colonne contenant les m-premiers vecteurs dominants orthonormalisés de la matrice A.

Question 5 : Nous pouvons utiliser les dimensions de H : Les couples propres de H correspondent au m premiers vecteurs propres dominants de A, donc le calcul des vecteurs propres de H suffit. La taille de H étant plus petite que celle de A, le calcul de ses éléments propres sera plus rapide que celui des éléments de A.

Question 6 :

```
27      % numéro de l'itération courante
28      k = 0;
29
30      % on génère un ensemble initial de m vecteurs orthogonaux
31      V = eye(n,m);
32
33      % rappel : conv = invariance du sous-espace V ||AV - VH||/||A|| <= eps
34      while (~conv & k < maxit),
35
36          k = k + 1;
37
38          % calcul de Y = A.V
39          Y = A*V;
40
41          % calcul de H, le quotient de Rayleigh H = V^T.A.V
42          H = V'*A*V;
43
44          % vérification de la convergence
45          acc = norm(Y - V*H, 'fro')/norm(A,'fro');
46          conv = (acc <= eps);
47
48          % orthonormalisation
49          V = mgs(Y);
50
51      end
```

```
52
53      % décomposition spectrale de H, le quotient de Rayleigh
54      [V1, V2] = eig(H);
55      V2 = diag(V2);
56
57      % on range les valeurs propres dans l'ordre décroissant
58      [V2,ind] = sort(V2, 'descend');
59      % on permute les vecteurs propres en conséquence
60      V1 = V1(ind);
61
62      % les m vecteurs propres dominants de A sont calculés à partir de ceux de H
63      vp = [];
64      for i = 1:m
65          vp = [vp V*V1(i)];
66      end
67      W = V2(1:m);
68
```

FIGURE 3 – Algorithme 2

Question 7 : Les étapes de l'algorithme 4 sont les suivantes (voir la figure 4).
L'initialisation de V et de k et de conv est faite des lignes 43 à 50.
La boucle TantQue est codée des lignes 52 à 110.
Les 3 premières opérations sont faite sur les lignes 55 à 59.
Puis la projection de Rayleigh-Ritz est à la ligne 62.
Enfin l'étape d'analyse de convergence a lieu des lignes 64 à 109.

```
43      % numéro de l'itération courante
44      k = 0;
45      % indicateur de la convergence
46      conv = 0;
47
48      % on génère un ensemble initial de m vecteurs orthogonaux
49      V = randn(n, m);
50      V = mgs(V);
51
52      % rappel : conv = (eigsum >= trace) | (nb_c == m)
53      while (~conv & k < maxit),
54
55          k = k+1;
56          %% Y <- A*V
57          Y = A*V;
58          %% orthogonalisation
59          V = mgs(Y);
60
61          %% Projection de Rayleigh-Ritz
62          [Wr, V] = rayleigh_ritz_projection(A, V);
63
64          %% Quels vecteurs ont convergé à cette itération
65          analyse_cvg_finie = 0;
66          % nombre de vecteurs ayant convergé à cette itération
67          nbc_k = 0;
68          % nb_c est le dernier vecteur à avoir convergé à l'itération précédente
69          i = nb_c + 1;
70
71          while(~analyse_cvg_finie),
72              % tous les vecteurs de notre sous-espace ont convergé
73              % on a fini (sans avoir obtenu le pourcentage)
74              if(i > m)
75                  analyse_cvg_finie = 1;
76              else
77                  % est-ce que le vecteur i a convergé
```

```
79                % calcul de la norme du résidu
80                aux = A*V(:,i) - Wr(i)*V(:,i);
81                res = sqrt(aux'*aux);
82
83                if(res >= eps*normA)
84                    % le vecteur i n'a pas convergé,
85                    % on sait que les vecteurs suivants n'auront pas convergé non plus
86                    % => itération finie
87                    analyse_cvg_finie = 1;
88                else
89                    % le_vecteur i a convergé
90                    % un de plus
91                    nbc_k = nbc_k + 1;
92                    % on le stocke ainsi que sa valeur propre
93                    W(i) = Wr(i);
94
95                    itv(i) = k;
96
97                    % on met à jour la somme des valeurs propres
98                    eigsum = eigsum + W(i);
99
100                   % si cette valeur propre permet d'atteindre le pourcentage
101                   % on a fini
102                   if(eigsum >= vtrace)
103                       analyse_cvg_finie = 1;
104                   else
105                       % on passe au vecteur suivant
106                       i = i + 1;
107                   end
108               end
109           end
110       end
```

FIGURE 4 – Algorithme 4

Question 8 : La matrice A^p est de taille n×n donc le calcul nécessite $2n^3p$ opérations. Pour calculer $A^p \times V$, il faut $2n^3p + 2n^2m$ opérations. Pour réduire le coût, on peut utiliser la récursivité : la matrice A×V est de taille n×m, donc le calcul de $A^p \times V$ demande moins d'opérations en procédant par récursivité, suivant le schéma A(A...A(AV).

Question 9 : Voir le script subspace-iter-v2.m.

Question 10 : Lorsque l'on augmente p, on observe que la méthode V2 est plus rapide que V0 et V1.

Question 11 : La précision n'est pas la même car avec la méthode V1, on continue de calculer

les vecteurs propres après avoir obtenu la convergence. Cela permet d'observer des valeurs des éléments propres plus précises qu'avec les autres méthodes.

Question 12 : Avec la méthode V3, on suppose que les calculs seront plus rapides car on bloque une partie des données, ce qui permet de manipuler des matrices de plus petites tailles et donc de diminuer le temps de calcul.

Question 13 : Voir le script subspace-iter-v3.m.

Question 14 : Les matrices (imat 1,2,3 et 4) ne possèdent pas le même spectre (VP), voir la figure 5.

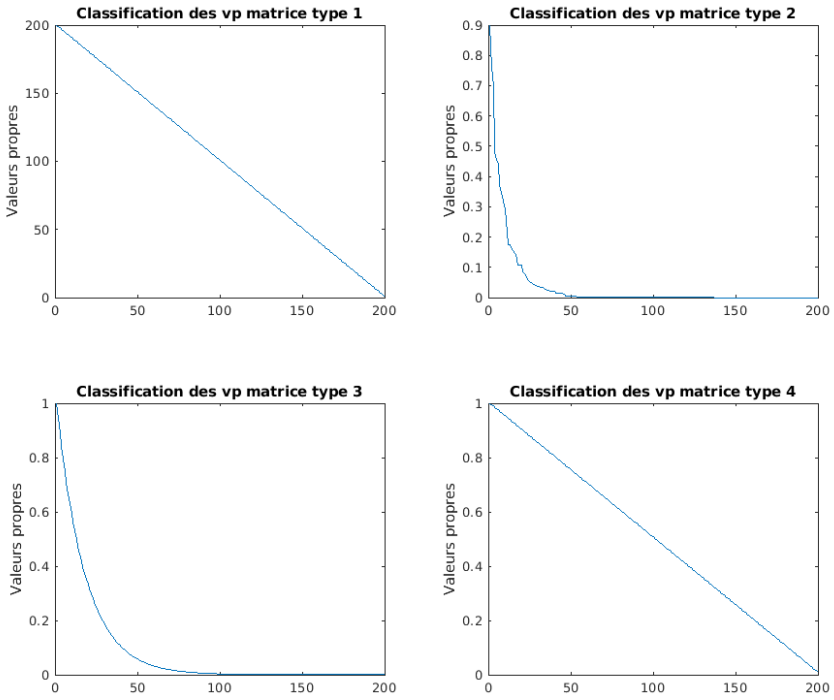

FIGURE 5 – Spectres des différents types de matrice

Question 15 : Pourle 1er et le 4e type de matrice, la méthode subspace-iter-v3 est plus rapide que les autres méthodes.
Pour le 2e et le 3e types, on observe les mêmes résultats, sauf pour le 2e type où subspace-iter-v2 est plus rapide.
Pour une matrice de petite taille, toutes les méthodes ont quasiment les mêmes performances avec un petit avantage pour la méthode power-v11.
Pour les méthodes subspace iteration on remarque l'amélioration en termes de performance au cours du projet. La version V3 avec l'approche en blocs et la déflation est supérieure aux autres(V2,V1 et V0) et s'approche des performances de "eig".

2 Conclusion

Cette deuxième partie du projet nous a permis de tester l'efficacité de la méthode de la puissance itérée aved déflation et ses améliorations et les méthodes subspace iteration.