TP5 Calcul Matriciel Numérique

Vernet Elliot, Imperatore Valentin, Courtois Thibault

January 2023

1 Introduction

On sait déjà calculer les valeurs propres d'une matrice donnée en utilisant son polynome caractéristique. Cependant, cette méthode est inutilisable pour des systèmes matriciels de taille supérieur à 5.
L'objectif de ce TP est d'utiliser trois nouvelles méthodes itératives : la méthode de puissance, la méthode de puissance inverse et la méthode de déflation pour déterminer des approximations des valeurs propres de matrices de taille importante.

2 Méthode de puissance

Soit une matrice A définie positive et $|\lambda_1|$, ... ,$|\lambda_n|$ ses valeurs propres dans l'odre décroissant.
On définit la suite de vecteur x_n telle que $x_{n+1} = Ax_n$ avec x_0 non orthogonale à e_1 le vecteur propre associé à la valeur propre λ_1.

Ainsi, $\frac{x_n}{||x_n||}$ converge vers un multiple de e_1.

Et $\frac{\langle x_{n+1}, x_n \rangle}{||x_n||^2}$ converge vers λ_1

Donc au bout d'un certain nombre d'itération, on peut approximer
$|\lambda_1| \approx \frac{x_{k+1}}{||x_k||}$ avec $x_k = A^k x_0$

C'est la méthode de puissance itérative. Elle permet de déterminer la valeur propre de plus grand module.

On va appliquer cette méthode à une matrice

$A = \begin{pmatrix} 5-\frac{3a}{4} & 2-\frac{3a}{4} & \frac{9a}{4}-8 \\ \frac{a+3}{4} & \frac{a+3}{4} & -\frac{3a+1}{4} \\ \frac{9-2a}{4} & \frac{5-2a}{4} & \frac{6a-15}{4} \end{pmatrix}$ avec a un réel.

On prend $x_0 = (1, 2, 3)^T$ comme vecteur d'initialisation.

On calcul dans un premier temps, les valeurs propres et les vecteurs propres de la matrice A pour différentes de valeur a avec la fonction eigh.

On obtient les résultats suivants :

a	valeurs propres	vecteurs propres
-100	-101, 2, 1	$\begin{pmatrix} 0.8018 \\ -0.2673 \\ 0.5345 \end{pmatrix}$, $\begin{pmatrix} -0.8165 \\ -0.4082 \\ -0.4082 \end{pmatrix}$, $\begin{pmatrix} 0.4082 \\ 0.8165 \\ 0.4082 \end{pmatrix}$
-2	-3, -2, 1	$\begin{pmatrix} -0.8018 \\ 0.2673 \\ -0.5345 \end{pmatrix}$, $\begin{pmatrix} 0.8165 \\ 0.4082 \\ 0.4082 \end{pmatrix}$, $\begin{pmatrix} 0.4082 \\ 0.8165 \\ 0.4082 \end{pmatrix}$
-1.5	-2.5, 2, 1	$\begin{pmatrix} -0.8018 \\ 0.2673 \\ -0.5345 \end{pmatrix}$, $\begin{pmatrix} 0.8165 \\ 0.4082 \\ 0.4082 \end{pmatrix}$, $\begin{pmatrix} 0.4082 \\ 0.8165 \\ 0.4082 \end{pmatrix}$
-1.1	-2.1, 2, 1	$\begin{pmatrix} -0.8018 \\ 0.2673 \\ -0.5345 \end{pmatrix}$, $\begin{pmatrix} 0.8165 \\ 0.4082 \\ 0.4082 \end{pmatrix}$, $\begin{pmatrix} 0.4082 \\ 0.8165 \\ 0.4082 \end{pmatrix}$
-1.001	-2.001, 2, 1	$\begin{pmatrix} -0.8018 \\ 0.2673 \\ -0.5345 \end{pmatrix}$, $\begin{pmatrix} 0.8165 \\ 0.4082 \\ 0.4082 \end{pmatrix}$, $\begin{pmatrix} 0.4082 \\ 0.8165 \\ 0.4082 \end{pmatrix}$

On va maintenant appliquer 10 itérations de la méthode de puissance et comparer les résultats avec ceux trouvés précédemment.

Code utilisé :

```
x0 =[1; 2; 3];
x = x0;
N=10;
for i = 1:N
    y=A*x;
    a=y(1)/x(1);
    b=y(2)/x(2);
    erreur = abs(a-b);
    x=y/norm(y);
end
```

Puis on va utiliser la méthode pour obtenir une précision de 10^{-10} sur la plus grande valeur propre, avec le code suivant :

```
x0 =[1; 2; 3];
x = x0;
N=10;
for i = 1:N
    y=A*x;
    a=y(1)/x(1);
    b=y(2)/x(2);
    erreur = abs(a-b);
    x=y/norm(y);
end
```

On en déduit le tableau de résultat suivant :

a	eigh	10 itérations	erreur
-100	$\begin{pmatrix} 0.8018 \\ -0.2673 \\ 0.5345 \end{pmatrix}$	$\begin{pmatrix} 0.8018 \\ -0.2673 \\ 0.5345 \end{pmatrix}$	2.1E-13
-2	$\begin{pmatrix} -0.8018 \\ 0.2673 \\ -0.5345 \end{pmatrix}$	$\begin{pmatrix} 0.7961 \\ -0.2838 \\ 0.5344 \end{pmatrix}$	0.5163
-1.5	$\begin{pmatrix} -0.8018 \\ 0.2673 \\ -0.5345 \end{pmatrix}$	$\begin{pmatrix} 0.7580 \\ -0.3792 \\ 0.5307 \end{pmatrix}$	2.8188
-1.1	$\begin{pmatrix} -0.8018 \\ 0.2673 \\ -0.5345 \end{pmatrix}$	$\begin{pmatrix} 0.0546 \\ -0.9722 \\ 0.2277 \end{pmatrix}$	10.2889
-1.001	$\begin{pmatrix} -0.8018 \\ 0.2673 \\ -0.5345 \end{pmatrix}$	$\begin{pmatrix} -0.4418 \\ -0.8930 \\ -0.0860 \end{pmatrix}$	4.6225

On remarque que plus a se rapproche de -1 et plus l'erreur augmente.

Maintenant on va comparer eigh et la méthode de puissance itérée jusqu'à la précision 10^{-10} :

a	λ_1 (eigh)	λ_1 (algo puissance)	nombre d'itérations
-100	-101	-101	1
-2	-3	-3	56
-1.5	-2.5	-2.5	108
-1.1	-2.1	-2.1	520
-1.001	-2.001	-2.001	51540

Ainsi, le nombre d'itérations de la méthode augmente significativement quand a se rapproche de -1 mais la méthode converge dans tous les cas testés.

3 Méthode de puissance inverse

La méthode de puissance itérative permet de déterminer la valeur propre de plus petit module. Elle consiste à appliquer la mérhode de puissance itérative à l'inverse de la matrice.

La suite x_n est alors définie tel que : $x_{n+1} = A^{-1}x_n$. On trouvera alors $\frac{1}{\lambda_n}$ car $\frac{1}{|\lambda_1|}$, ... ,$\frac{1}{|\lambda_n|}$ est ici dans l'odre décroissant.
On va utiliser la même matrice A que pour la partie précédente.
On commence par trouver les éléments propres de la matrice avec la fonction eigh sur Matlab.

On en déduit le tableau de résultats suivant :

a	valeurs propres	vecteurs propres
0.95	2, 1, -0.05	$\begin{pmatrix}0.8165\\0.4082\\0.4082\end{pmatrix}$, $\begin{pmatrix}0.4082\\0.8165\\0.4082\end{pmatrix}$, $\begin{pmatrix}0.8018\\-0.2673\\0.5345\end{pmatrix}$
0.5	2, 1, -0.5	$\begin{pmatrix}-0.8165\\-0.4082\\-0.4082\end{pmatrix}$, $\begin{pmatrix}0.4082\\0.8165\\0.4082\end{pmatrix}$, $\begin{pmatrix}0.8018\\-2.673\\0.5345\end{pmatrix}$
0.1	2, 1, -0.9	$\begin{pmatrix}0.8165\\0.4082\\0.4082\end{pmatrix}$, $\begin{pmatrix}0.4082\\0.8165\\0.4082\end{pmatrix}$, $\begin{pmatrix}0.8018\\-0.2673\\0.5345\end{pmatrix}$
0.5	2, 1, -0.5	$\begin{pmatrix}0.8165\\0.4082\\0.4082\end{pmatrix}$, $\begin{pmatrix}-0.4082\\-0.8165\\-0.4082\end{pmatrix}$, $\begin{pmatrix}0.8018\\-0.2673\\0.5345\end{pmatrix}$
0.001	2, 1, -0.9990	$\begin{pmatrix}0.8165\\0.4082\\0.4082\end{pmatrix}$, $\begin{pmatrix}-0.4082\\-0.8165\\-0.4082\end{pmatrix}$, $\begin{pmatrix}0.8018\\-2.673\\0.5345\end{pmatrix}$
0	2, -1, 1	$\begin{pmatrix}0.8165\\0.4082\\0.4082\end{pmatrix}$, $\begin{pmatrix}0.8018\\-0.2673\\0.5345\end{pmatrix}$, $\begin{pmatrix}-0.4082\\-0.8165\\-0.4082\end{pmatrix}$

On code l'algorithme de puissance inverse sur matlab pour 10 itérations puis jusqu'à une précision donnée.

On obtient les programmes suivants :

```
x0 =[1; 2; 3];
x = x0;
N=10;
for i = 1:N
    y=A*x;
    a=y(1)/x(1);
    b=y(2)/x(2);
    erreur = abs(a-b);
    x=y/norm(y);
end
```

```
E = 10^6;
eps = 10^-10;
iteration = 0;
while E > eps
    y=A*x;
    a=y(1)/x(1);
    b=y(2)/x(2);
    E = abs(a-b);
    x=y/norm(y);
    iteration = iteration + 1;
end
```

On obtient à partir de ces programmes les résultats suivants :

a	0.95	0.5	0.1	0.001	0
erreur	5.57E-13	0.008	4.5278	8.2393	8.5111

On en déduit que plus a diminue vers 0 et plus l'erreur augmente.

a	λ_3 (eigh)	λ_3 (algo puissance)	nombre d'itérations
0.95	-0.05	-0.05	1
0.5	-0.5	-0.5	27
0.1	-0.9	-0.9	231
0.001	-0.999	-0.999	25391
0	1	?	∞

La valeur de λ_3 dans la cas a = 0 n'a pas été trouvée. Autrement, la méthode converge vers la bonne valeur et à besoin de plus d'itération quand la valeur de a tend vers 0.

4 Méthode de déflation

On considère toujours que : $|\lambda_1|$, ... ,$|\lambda_n|$ est dans l'odre décroissant.
Soit A une matrice définie positive dont on veut trouver les valeurs propres (toutes).
On applique une premère fois la méthode de puissance itérative. On obtient alors λ_1.

On note $(e_1, ..., e_n)$ une base orthonormée de vecteur propre associés aux valeurs propres de la matrice A.
On pose $B = A - \lambda_1 e_1 e_1^T$

Alors B est une matrice qui a pour valeurs propres $0, \lambda_2, ..., \lambda_n$.

On peut donc réutiliser la méthode de puissance sur la matrice B et trouver la valeur propre λ_2.
Il suffit de réitérer se procéder pour trouver l'ensemble des valeurs propres de A.

On va effectuer les mêmes tests que dans les parties précédentes. Cependant, la méthode va nous permettre de trouver une approximation des trois valeurs propres de la matrice A.

Pour 10 itérations, le programme utlisée pour la méthode de défkation est le suivant :

```
x0 =[1; 2; 3];
x = x0;
N=10;
for i = 1:N
    y=A*x;
    a=y(1)/x(1);
    b=y(2)/x(2);
    erreur = abs(a-b);
    x=y/norm(y);
end
```

On obtient les résultats suivants :

a	0.95	0.5	0.1	0.001	0
erreur	1.56E-10	0.0205	2.472	3.6822	3.6305

Encore une fois, l'erreur augmente quand on rapproche a de 0.

Pour arriver à une précision donner, on utilise le code suivant :

```
x0 =[1; 2; 3];
x = x0;
N=10;
e1 = [vect(1, 1), vect(2, 1), vect(3, 1)];
B = A - val(1, 1)*e1'*e1;
E = 10^-6;
eps = 10^-10;
iteration = 0;
while E>eps
    y = B*x;
    a = y(1)/x(1);
    b = y(2)/x(2);
    E = abs(a-b);
    x = y/norm(y);
    iteration = iteration + 1;
end
iteration
x'*B*x
```

On en déduit les résultats ci-dessous :

a	λ_2 (eigh)	λ_2(méthode de déflation)	nombre d'itérations
0.95	1	1	11
0.5	1	1	38
0.1	1	1	238
0.001	1	1	24847
0	1	?	∞