Université Lyon 1 Année 2019-2020
Licence de Mathématiques, 3e année
Analyse matricielle

Corrigé du TP 2

Exercice 1. (*Suites récurrentes*)

Soit $n \in \mathbb{N}^*$, $B \in \mathcal{M}_n(\mathbb{R})$ et $b \in \mathbb{R}^n$. On définit une suite récurrente associée à la matrice d'itération B et au vecteur b par

$$\begin{cases} x_0, x_1 \in \mathbb{R}^n \text{ donnés} \\ x_{k+1} = Bx_k + Bx_{k-1} + b \text{ pour tout } k \in \mathbb{N}^*. \end{cases}$$

On rappelle (cf. examen partiel du 15 mars 2019) que la suite $(x_k)_{k\in\mathbb{N}}$ converge, quels que soient x_0 et x_1, si et seulement si $\rho(C) < 1$ où

$$C = \begin{pmatrix} B & B \\ I_3 & 0 \end{pmatrix}$$

1. On considère la matrice et les vecteurs

$$B = \frac{1}{20}\begin{pmatrix} 1 & 2 & 3 \\ 2 & 0 & 2 \\ 3 & 4 & 5 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad u^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad u^{(1)} = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

et la suite de vecteurs définie par

$$\forall n \in \mathbb{N}^*, \quad u^{(n+1)} = Bu^{(n)} + Bu^{(n-1)} + b. \tag{1}$$

(a) Calculer les premiers termes de la suite $(u^{(n)})_n$. On pourra utiliser `print("%.20f",...)` pour que les nombres soient affichés sous python avec 20 chiffres après la virgule. Qu'observe-t-on ?

RÉPONSE :

$$u^{(2)} = \begin{pmatrix} 1.9 \\ 1.6 \\ 2.8 \end{pmatrix}, \quad u^{(3)} = \begin{pmatrix} 2.275 \\ 1.87 \\ 3.505 \end{pmatrix}, \quad u^{(4)} = \begin{pmatrix} 2.5015 \\ 2.048 \\ 3.8965 \end{pmatrix}, \quad u^{(5)} = \begin{pmatrix} 2.74085 \\ 2.2178 \\ 4.35045 \end{pmatrix}$$

En calculant une centaine d'itérations, on observe que la suite semble tendre vers une limite.

(b) Quel rayon spectral doit-on calculer pour justifier la convergence ? Déterminer sa valeur.

RÉPONSE :
Pour justifier la convergence de la suite il nous faudra calculer le rayon spectral de la matrice

$$C = \begin{pmatrix} B & B \\ I_3 & 0 \end{pmatrix} = \frac{1}{20} \begin{pmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 2 & 0 & 2 & 2 & 0 & 2 \\ 3 & 4 & 5 & 3 & 4 & 5 \\ 20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 & 0 \end{pmatrix}.$$

On utilise la commande `E1, V1 = npl.linalg.eig(C)` où `E1` contiendra des valeurs approchées des valeurs propres de C et `V1` des valeurs approchées des vecteurs propres de C.
Ainsi une valeur approchée du rayon spectral de la matrice C est donnée par `rho=max(np.abs(E1))= 0.87018538430899.`

(c) Si elle converge, quelle est la limite de la suite $(u^{(n)})_n$? Justifier la réponse.

RÉPONSE :
Le rayon spectral de la matrice C étant strictement inférieur à 1, la suite converge (cf rappel en début d'exercice).
Notons u sa limite. En passant à la limite dans la relation récurrence (1), on obtient $u = Bu + Bu + b$, u est donc la solution de $(I_3 - 2B)u = b$.
Numériquement, on obtient une valeur approchée de u :

$$\lim_{n\to\infty} u^{(n)} = u \simeq \begin{pmatrix} 4.22413793 \\ 3.27586207 \\ 7.15517241 \end{pmatrix}.$$

2. Répondre aux mêmes questions avec

$$B = \frac{1}{10} \begin{pmatrix} 1 & 2 & 3 \\ 2 & 0 & 2 \\ 3 & 4 & 5 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad u^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad u^{(1)} = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

RÉPONSES :

(a)

$$u^{(2)} = \begin{pmatrix} 5.195515 \\ 3.94365 \\ 9.05251 \end{pmatrix}, \; u^{(3)} = \begin{pmatrix} 7.196773 \\ 5.375004 \\ 12.830296 \end{pmatrix}, \; u^{(4)} = \begin{pmatrix} 10.66780 \\ 7.855020 \\ 19.3865593 \end{pmatrix}, \; u^{(5)} = \begin{pmatrix} 15.0975197 \\ 11.0162870 \\ 27.7598119 \end{pmatrix}$$

On calcule une centaine de termes, on observe que la suite ne semble pas converger (il semble que la norme de $u^{(n)}$ tende vers l'infini).

(b) Pour étudier la convergence de la suite on calcule le rayon spectral de la matrice

$$C = \begin{pmatrix} B & B \\ I_3 & 0 \end{pmatrix} = \frac{1}{10} \begin{pmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 2 & 0 & 2 & 2 & 0 & 2 \\ 3 & 4 & 5 & 3 & 4 & 5 \\ 20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 & 0 \end{pmatrix}.$$

On procède comme à la question précédente et on obtient une valeur approchée de

$$\rho(C) \simeq 1.391664648116025.$$

(c) Le rayon spectral de la matrice de l'itération étant supérieur à 1, la suite diverge.

Exercice 2. (*Calcul de valeurs propres*)

Méthode de la puissance itérée. On rappelle qu'il s'agit d'une méthode itérative très simple permettant de calculer une approximation de la valeur propre de plus grand module (lorsqu'il n'y en a qu'une qui ait ce module) d'une matrice A et un vecteur propre associé.

Une implémentation sous python de cette fonction est donnée ci-dessous.

```
def puissance(A,z0,tol,nitermax):
    q = z0/npl.norm(z0)
    q2 = q
    err = []
    nu1 = []
    res = tol + 1
    niter = 0
    z = np.dot(A,q)
    while ((res > tol) and (niter <= nitermax)):
        q = z/npl.norm(z)
        z = np.dot(A,q)
        lam = np.dot(q,z)
        x1 = q
        z2 = np.dot(q2,A)
        q2 = z2/npl.norm(z2)
        y1 = q2
```

```
    c = np.dot(y1,x1)
    if c > 5E-2:
        res = npl.norm(z - lam*q)/c
        niter = niter + 1
        err.append(res)
        nu1.append(lam)
    else:
        print("Problème de convergence !")
        break
return(nu1,x1,niter,err)
```

Les valeurs en entrée `z0`, `tol` et `nitermax` sont respectivement le vecteur initial, la tolérance pour le test d'arrêt et le nombre maximum d'itérations admissible. En sortie, `x1` et `niter` sont respectivement les approximations du vecteur propre unitaire x_1 et le nombre d'itérations nécessaire à la convergence de l'algorithme, le vecteur `nu1` contient les approximations successives de la valeur propre cherchée, tandis que `err` contient la taille des résidus successifs.

1. Programmer la fonction `puissance`.

2. On considère la matrice
$$A = \begin{pmatrix} 15 & -1 & 1 \\ 1 & 9 & -2 \\ -2 & 2 & 0 \end{pmatrix}$$
   (a) Utiliser la fonction `puissance` pour rechercher la valeur propre dominante de A, ainsi qu'un vecteur propre associé en prenant le vecteur $z_0 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T / \sqrt{3}$ comme vecteur initial, et une tolérance égale à 10^{-8} pour le critère d'arrêt.
   (b) Valider le résultat en utilisant la commande `eigvals` dans `numpy.linalg` de python.

RÉPONSES :

(a) À l'aide de `nu1, x1, niter, err=puissance(A, z0, 1e-8, 100)` on peut obtenir des valeurs approchées de la valeur propre dominante ν de A et d'un vecteur propre v associé en utilisant `nu1` et `x1`. Attention, `nu1` est un vecteur, on doit prendre sa dernière composante. On obtient
$$\nu \simeq 14.679074138653458, \quad v \simeq \begin{pmatrix} 0.97257666 \\ 0.20794521 \\ -0.10417979 \end{pmatrix}$$

(b) En guise de validation on utilise les fonctions préprogrammées, la commande
```
nu2=np.linalg.eigvals(A)
print(np.amax(nu2))
```
donne la valeur approchée suivante pour ν : $\nu \simeq 14.679074141287877$ proche de celle obtenue à la question précédente à 10^{-7} près.

3. On veut évaluer la vitesse de convergence de la méthode. Pour cela, on considère, pour $a \in \mathbb{R}$, la matrice
$$A = \begin{pmatrix} 1 & a & a \\ a & 1 & a \\ a & a & 1 \end{pmatrix}$$

dont on rappelle qu'elle est diagonalisable et a pour valeurs propres $1-a$ (double) et $1+2a$.

(a) Tester la méthode avec par exemple un vecteur z_0 choisi aléatoirement et tracer, sur une même figure, le logarithme de la norme du résidu en fonction du numéro d'itération, ainsi que la droite passant par 0 et de pente $(\log|\lambda_2| - \log|\lambda_1|)$ où λ_1 est la valeur propre de plus grand module et λ_2 l'autre valeur propre, pour différentes valeurs de $a \in]0,1[$. Pour la valeur $a = 1/2$, donner un titre (commande `plt.title()`), une légende à chaque courbe (commande `plt.legend()`) et nommer les axes (commandes `plt.xlabel()`, `plt.ylabel()`), et
↪ *Sauvegarder la figure dans un fichier* `figure1.jpg` (au format jpg).
Qu'observe-t-on ? Pourquoi ?
RÉPONSE :
Pour différentes valeurs de a on obtient les courbes suivantes.

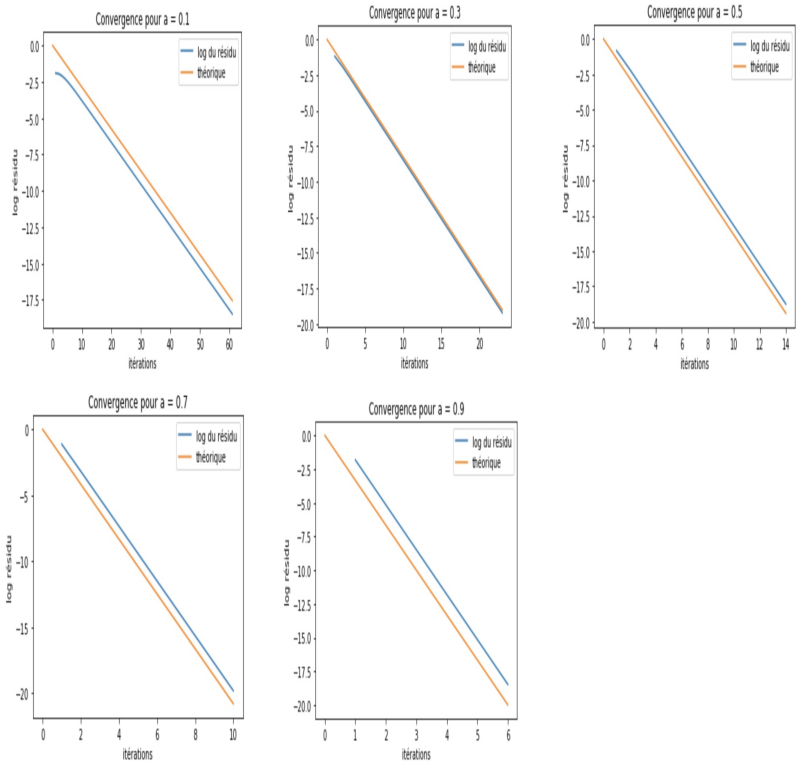

FIGURE 1 – Test vitesse de convergence de la méthode de la puissance

Il a été vu en cours que, pour tout k, le résidu d'ordre k est majoré par $C\left|\frac{\lambda_2}{\lambda_1}\right|^k$ (où λ_1 et λ_2 sont la plus grande et la deuxième plus grande valeurs propres en module).
Notons $r^{(k)}$ le résidu d'ordre k, si on avait une égalité : pour tout k, $r^{(k)} = C\left|\frac{\lambda_2}{\lambda_1}\right|^k$, on aurait $\ln r^{(k)} = \ln C + k \ln\left|\frac{\lambda_2}{\lambda_1}\right|$.
On observe bien des droites sur les différentes sous-figures de la figure 1, de pente $(\log|\lambda_2| - \log|\lambda_1|) = \log(\frac{|1-a|}{|1+2a|})$.
Numériquement, on peut aussi obtenir la pente à l'aide d'une régression linéaire.

(b) Faire la même chose pour $a = -1$.
↪ *Sauvegarder la figure dans un fichier* `figure2.jpg` (au format jpg).
Le théorème vu en cours permet-il de conclure à la convergence de la méthode dans ce cas ? Qu'observe-t-on ?

RÉPONSE :
Pour la valeur de $a = -1$ on obtient la courbe suivante.

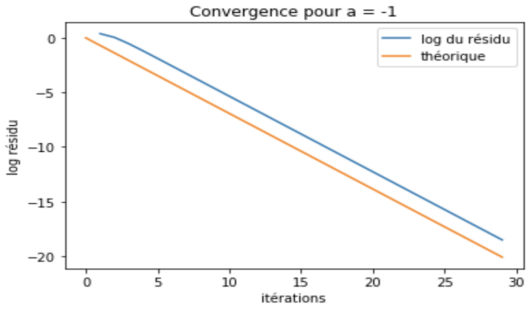

FIGURE 2 – Test vitesse de convergence de la méthode de la puissance pour $a = -1$

Le théorème vu en cours ne permet pas de conclure sur la convergence de la méthode car pour $a = -1$ la valeur propre de plus grand module est $1 - a = 2$, elle n'est pas simple (elle est double). Pourtant on observe encore la convergence de la méthode, avec la même vitesse de convergence.
On peut montrer un théorème plus général de convergence de la méthode de la puissance. L'hypothèse fondamentale assurant la convergence est qu'il n'y ait qu'une valeur propre de plus grand module, mais celle-ci peut être multiple. Selon les cas, cela modifie la vitesse de convergence mais pas le fait que la méthode converge. On va le voir sur l'exemple suivant.

4. On considère maintenant la matrice

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Le théorème vu en cours permet-il de conclure à la convergence de la méthode dans ce cas ? Pourquoi ?

RÉPONSE : Comme précédemment, le théorème du cours ne permet pas de conclure quant à la convergence car A a une seule valeur propre qui est double.

En initialisant avec $z_0 = (1, 1)$, tester la méthode de la puissance et tracer, sur une même figure, le logarithme de la norme du résidu en fonction du *logarithme* du numéro d'itération, ainsi que la droite passant par 0 et de pente -1. Donner un titre et nommer les axes, et ↪ *Sauvegarder la figure dans un fichier* `figure3.jpg` (au format jpg).
RÉPONSE :
On obtient la courbe suivante :

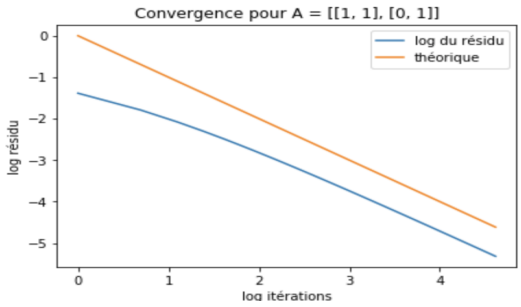

FIGURE 3 – méthode de la puissance de la matrice A avec vecteur initiale $z_0 = (1, 1)$

Quel semble être le comportement du résidu en fonction du numéro d'itération ?

RÉPONSE :
Si on note encore $r^{(k)}$ le résidu d'ordre k, la figure obtenue semble donner : pour tout k, $\ln r^{(k)} = c - \ln k$ c'est-à-dire que $r^{(k)} = \frac{\tilde{c}}{k}$.
On observe encore une convergence mais beaucoup moins rapide que précédemment.

Méthode QR et calcul de valeurs propres. On rappelle que lorsqu'une matrice A a des valeurs propres de modules tous distincts, les matrices de la suite (T^k) définie par la récurrence

$$\begin{aligned} &T^0 = A \\ &Q^k R^k = T^{(k-1)} \quad \text{(décomposition QR de } T^{(k-1)}\text{)} \\ &T^k = R^k Q^k \end{aligned}$$

voient leurs termes sous-diagonaux converger vers 0, leurs termes sur-diagonaux rester bornés et leurs termes diagonaux converger vers les valeurs propres de A. Si A est symétrique alors chacune de ces matrices est symétrique, et elles convergent vers une matrice diagonale.

La commande `qr` dans `numpy.linalg` de python contient une implémentation de la décomposition QR. On peut alors écrire une implémentation directe de la méthode QR utilisant cette fonction de la manière suivante

```
def methode_qr(A,niter):
    T = A
    for i in range(niter):
        Q,R = npl.qr(T);
        T = np.dot(R,Q);
    return(T,Q,R)
```

où `niter` est le nombre d'itérations souhaité.

1. Programmer la fonction `methode_qr`.

2. Construire une matrice symétrique A de taille 4×4 telle que $a_{ij} = 4+i-j$ pour $1 \leq i \leq j \leq 4$.

RÉPONSE :

$$A = \begin{pmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

3. Vérifier que la matrice T^{20} obtenue après 20 itérations de la méthode QR (avec la matrice A de la question 2) est « presque » diagonale (les éléments non diagonaux sont presque nuls).

RÉPONSE : Effectivement la matrice T^{20} obtenue ci-dessous est presque diagonale

$$T^{20} = \begin{pmatrix} 11.0990195 & 6.44596954e-10 & -1.37447950e-16 & 1.22398079e-15 \\ 6.44596522e-10 & 3.41421356 & 1.42862490e-11 & 2.13092814e-15 \\ 1.74359452e-21 & 1.42861613e-11 & 0.900980444 & 1.16034869e-04 \\ 2.31847597e-25 & 2.67693034e-15 & 1.16034869e-04 & 0.585786480 \end{pmatrix}$$

4. Utiliser la fonction `npl.eigvals` pour vérifier les valeurs propres obtenues.

RÉPONSE : On trouve les valeurs propres suivantes pour A

11.09901951; 3.41421356; 0.90098049; 0.58578644

5. Programmer la version suivante (vue en cours) de l'algorithme global de recherche de valeurs propres.

```
def methode_qr_2(A,niter):
    Z = A
    for i in range(niter):
        Q,R = npl.qr(Z)
        Z = np.dot(A,Q)
    T = np.dot(np.dot(np.transpose(Q),A),Q)
    return(T,Q,R)
```

où `niter` est le nombre d'itérations souhaité. Utiliser cette fonction pour la même matrice A de la question 2, avec 20 itérations. Vérifier que les colonnes de Q sont des approximations des vecteurs propres de A (ceci est dû à la symétrie de A). Quel est l'avantage de cette version de l'algorithme ?

RÉPONSE : On obtient d'une part la matrice

$$Q = \begin{pmatrix} -0.44829785 & -0.65328148 & 0.54693505 & -0.27039672 \\ -0.54683547 & -0.27059805 & -0.44853832 & 0.6531164 \\ -0.54683547 & 0.27059805 & -0.44805733 & -0.65344647 \\ -0.44829785 & 0.65328148 & 0.54673582 & 0.27079934 \end{pmatrix}$$

D'autre part les vecteurs propres de A

$$v_1 = \begin{pmatrix} 0.44829785 \\ 0.54683547 \\ 0.54683547 \\ 0.44829785 \end{pmatrix} ; v_2 = \begin{pmatrix} 0.65328148 \\ 0.27059805 \\ -0.27059805 \\ -0.65328148 \end{pmatrix} ; v_3 = \begin{pmatrix} 0.54683547 \\ -0.44829785 \\ -0.44829785 \\ 0.54683547 \end{pmatrix} ; v_4 = \begin{pmatrix} -0.27059805 \\ 0.65328148 \\ -0.65328148 \\ 0.27059805 \end{pmatrix}$$

Ce qui montre que les colonnes de Q sont des approximations des vecteurs propres de A.

Avantage : c'est facile à coder et ça fournit, en plus de l'approximation des valeurs propres, l'approximation des vecteurs propres associés à travers les colonnes de la matrice sortie, Q, de la méthode.