Feuille de TP numéro 6

Exercice 1 : **Calcul des valeurs propres**

A l'aide de `rand` du package `numpy.random`, déterminer U matrice aléatoire dans $\mathcal{M}_5(\mathbb{R})$ et pour `B=diag([2,-3,-1,0.5,-5.0])`. On pose $A = UBU^{-1}$. On pourra commencer par fixer le générateur des nombres aléatoires à l'aide de `seed` du package `numpy.random`.

1) Calculer A et utiliser la fonction `eig` du module `scipy.linalg` pour calculer une approximation des éléments propres de A. Stocker et afficher les valeurs propres.
On pourra utiliser `argsort` du module `numpy` pour classer les valeurs propres ainsi obtenues dans l'ordre croissant des modules.

2) La méthode QR calcule une suite de matrices orthogonalement semblables à A qui converge sous certaines conditions vers une matrice triangulaire supérieure. Elle consiste à poser $T^{(0)} = A$ puis à calculer $T^{(k+1)} = R^{(k)}Q^{(k)}$ où les matrices $Q^{(k)}$ et $R^{(k)}$ sont obtenues par décomposition QR de $T^{(k)}$ i.e. $T^{(k)} = Q^{(k)}R^{(k)}$.
On utilisera la fonction `qr` du module `scipy.linalg`.

a) Mettre en oeuvre le processus itératif : on arrêtera le processus quand le maximum des modules des termes sous-diagonaux des itérés de la méthode QR (obtenu avec `tril(T,-1)` de `numpy`) sera plus petit que 10^{-10}.
On affichera à chaque itération avec un format le numéro de l'itéré et le maximum des modules des termes sous-diagonaux des itérés.

b) A convergence, afficher la matrice obtenue et commenter le résultat. Qu'a-t-il de remarquable ?

Compléments mathématiques :
Les matrices, D. Serre. / *Analyse numérique matricielle*, L. Amodei, J.-P. Dedieu. / *Introduction to numerical analysis*, J. Stoer, R. Bulirsch

- Les matrices $T^{(k)}$ sont toutes semblables à A.
- Démontrer par récurrence que si on pose $P^{(k)} = Q^{(0)}Q^{(1)}\dots Q^{(k-1)}$ et si on pose $U^{(k)} = R^{(k-1)}\dots R^{(0)}$ alors $P^{(k)}U^{(k)}$ est la factorisation QR de A^k. La suite de la preuve s'appuie sur l'unicité de la factorisation QR.
- On peut démontrer le résultat suivant : Si $A \in Gl_n(\mathbb{C})$ admet des valeurs propres distinctes en module et si la matrice de passage à un système diagonal admet une décomposion LU alors la partie triangulaire inférieure stricte de $T^{(k)}$ tend vers 0 et sa diagonale tend vers le vecteur des valeurs propres rangées dans l'ordre décroissant des modules. (Remarque : ce théorème est technique, il n'est pas recommandé de le proposer en développement).
- Le taux de convergence des coefficients $T^k_{i,i-1}$ est fonction du rapport des modules des

valeurs propres.

4) Dans le package `scipy.linalg`, la fonction `hessenberg` transforme une matrice en une matrice semblable de la forme de Hessenberg. Appliquer cette fonction à A et reprendre la question 2) en appliquant la méthode QR à cette nouvelle matrice. Contrôler que les itérés de la méthode QR restent des matrices de Hessenberg.

Remarque : la factorisation QR d'une matrice de Hessenberg est moins coûteuse qu'une matrice pleine : n^2 opérations au lieu de n^3.

Remarque : *Analyse matricielle*, J.E. Rombaldi. Si A est une matrice symétrique, la fonction `hessenberg` transforme la matrice en une matrice semblable tridiagonale en utilisant des matrices de Householder. La méthode de Jacobi (mieux adaptée que QR dans le cas symétrique) qui utilise des rotations pour annuler les coefficients permet alors d'obtenir une approximation des éléments propres.

Exercice 2 : **Calcul de la plus grande valeur propre en module**

On considère la matrice définie dans l'exercice 1.

1) Calculer à l'aide de la méthode de la puissance la valeur propre de A de plus grand module. L'algorithme initialisé avec un vecteur aléatoire x de norme 1 s'écrit à chaque itération

$$y = Ax \qquad , \qquad \nu = \langle y, x \rangle \qquad , \qquad x = \frac{y}{||y||} \, ,$$

où $\langle \cdot, \cdot \rangle$ est le produit scalaire ou hermitien. Le test d'arrêt portera sur l'erreur relative : $\frac{|\nu_k - \nu_{k-1}|}{|\nu_k|} < 10^{-8}$. On prendra $\nu_0 = 1$. Afficher à chaque itération, le numéro de l'itération k, l'erreur relative et la valeur convergeant vers la valeur propre recherchée.

2) Sur un graphique, tracer en échelle semi-logarithmique
- — la différence relative,
- — la différence en module entre la valeur propre de plus grand module obtenue avec `eig` et la valeur convergeant vers la valeur propre recherchée,
- — le rapport $|\lambda_{n-1}/\lambda_n|^k$ où λ_n est la valeur propre de plus grand module et λ_{n-1} est la valeur propre de module immédiatement plus petit.

3) La méthode de la puissance inverse consiste à chercher la plus petite valeur propre en module de A en cherchant la plus grande valeur propre en module de A^{-1}. Reprendre et **adapter** les questions 1) et 2) à la méthode de la puissance inverse. On prendra le même test d'arrêt et on résoudra le système linéaire à l'aide de `solve` de `scipy.linalg`. On comparera

4) Montrer théoriquement et vérifier numériquement en adaptant les questions 1 et 2 que la méthode de la puissance inverse appliquée à $A - 3.5I$ (I étant la matrice identité) permet d'obtenir la valeur propre 2.0 de A, c'est-à-dire la valeur propre la plus proche de 3.5. On résoudra le système linéaire à l'aide de `solve` de `scipy.linalg`. Attention le décalage peut modifier l'ordre des valeurs propres "décalées" en module. 1

Remarque : Ce décalage permet non seulement de trouver une valeur propre dont on a une estimation mais cela permet également d'accélérer le processus itératif.

5) Reprendre la méthode de la puissance avec `B=diag([2,-3,-1,0.5,-5.0+1.0j])`. On adaptera la définition du produit scalaire pour le transformer en un produit sesquilinéaire.

Remarque On peut oberver des problèmes de convergence de la méthode de la puissance ou de la méthode QR lorsque A n'est pas diagonalisable.
Remarque On peut oberver des problèmes de convergence de la méthode QR lorsque la matrice possède plusieurs valeurs propres de plus grand module.

Exercice 3 : **Calcul des racines (toutes réelles) d'un polynôme**

Soit $P = 80x^7 + 44x^6 - 1372x^5 + 527x^4 + 4829x^3 - 5791x^2 + 1863x - 180$.

1) Utiliser la fonction `roots` du module `numpy` ou la méthode `r` de la classe `poly1d` de `numpy` pour calculer une approximation des racines de P.

2) Appliquer la méthode QR à la matrice compagnon A de P obtenue à l'aide de la fonction `companion` du module `scipy.linalg`. Le test d'arrêt portera sur le maximum des modules des coefficients sous diagonaux calculé avec l'aide de `tril`.

3) A convergence, comparer les racines de P et les termes diagonaux de la matrice triangulaire supérieure ainsi obtenue.

Remarque la fonction `roots` calcule les valeurs propres de la matrice compagnon pour déterminer les racines d'un polynôme.
Remarque : la fonction `roots` calcule les valeurs propres de la matrice compagnon pour déterminer les racines d'un polynôme avec la méthode QR. La matrice compagnon est une matrice de Hessenberg.

Exercice 4 : **Calcul des racines complexes d'un polynôme**

1) Soit $P = x^3 + 2x^2 + x + 2$ dont les racines sont -2, i et $-i$.

2) Calculer la matrice compagnon A de P. Appliquer la méthode QR à cette matrice. La méthode ne convergeant pas vers une matrice triangulaire supérieure, le test d'arrêt portera sur le module du coefficient sous diagonal : $|T_{21}^{(k)}| < 10^{-6}$. Afficher à chaque itération la matrice $T^{(k)}$. Qu'observe-t-on ?

3) Afficher également avec un format, le numéro de l'itéré, le module de T_{21} et une des 2 valeurs propres calculées avec `eig` de `scipy.linalg` de la matrice extraite de $T^{(k)}$ et constituée des 2 dernières lignes et colonnes de $T^{(k)}$.

Remarque : Au lieu d'utiliser `eig`, on pourrait résoudre le polynôme caractéristique de degré 2 de cette matrice de $\mathcal{M}_2(\mathbb{R})$,

Retrouver ainsi le résultat (6.6.4.15) de Stoer-Bulirsch, 2nd édition :
Soit A matrice réelle d'ordre n avec une paire de valeurs propres λ_r, λ_{r+1} complexes conjugués et $n-2$ valeurs propres réelles telles que

$$|\lambda_1| > \cdots > |\lambda_r| = |\lambda_{r+1}| > \cdots > |\lambda_n|$$

alors

- i) $\lim_{k \longrightarrow +\infty} T^k_{i,j} = 0$ pour tout $(i,j) \neq (r+1, r)$ avec $i > j$,
- ii) $\lim_{k \longrightarrow +\infty} T^k_{i,i} = \lambda_i$ pour $i \neq r, r+1$,
- iii) les matrices $\begin{pmatrix} T^k_{r,r} & T^k_{r,r+1} \\ T^k_{r+1,r} & T^k_{r+1,r+1} \end{pmatrix}$ divergent en général mais leurs valeurs propres convergent vers λ_r, λ_{r+1}.

Exercice 5 : **Calcul du facteur orthogonal de la décomposition polaire**

Soit $N = 20$ et A une matrice aléatoire de taille $N \times N$ obtenue avec la fonction `random.rand` du package `numpy`.

1) Calculer le facteur orthogonal $W = UV^T$ de A en utilisant la décomposition svd de la matrice $A = U\Sigma V^T$. On utilisera la fonction `svd` de `scipy.linalg` et on vérifiera le résultat de la décomposition (l'orthogonalité de U, l'othogonalité de V et la décomposition).

2) Mettre en oeuvre le processus itératif définie par $X_0 = A$ et $X_{k+1} = \frac{1}{2} X_k (I + (X_k^T X_k)^{-1})$. On utilisera `inv` de `scipy.linalg`. On arrêtera le processus quand $\|X_{k+1} - X_k\|_F < 10^{-7}$ ou quand $k > 15$. On affichera à chaque itération le numéro de l'itération, la norme $\|X_{k+1} - X_k\|_F$ et la norme $\|W - X_k\|_F$ avec un format de façon à mettre en évidence la convergence quadratique. La norme de Frobenius est paramétrée dans la fonction `norm` de `scipy.linalg`.

3) Stocker dans un tableau les valeurs $\|X_{k+1} - X_k\|_F$ au cours des itérations. Tracer en échelle logarithmique $\|X_{k+1} - X_k\|_F$ en fonction de $\|X_k - X_{k-1}\|_F$. Mettre en évidence la pente.