Projet TAN2

Pierre-Antoine Lambrecht

April 2023

1 Diagonalisation

Avant toutes choses on importe les librairies nécessaires au projet en leur donnant des alias pour rendre leurs utilisations plus agréable.

On crée la matrice A avec *numpy* ce qui permet de manipuler les matrices plus facilement et peut être aussi plus efficassement qu'avec un simple tableau python de base de la forme $A = []$.

Pour diagonaliser la matrice A il va falloir :

1. vérifier que A est **diagonalisable**
2. calculer les **valeurs propres** de A
3. calculer un **vecteur propre** associé à chaque valeur propre

On utilise donc la commande *eig* de la librairie *numpy.linalg* (pour linear algebra) qui prend comme argument une matrice et renvoit un couple de vecteurs (valeurs propre, vecteur propre). Comme les valeurs propre sont distincts, χ_A le polynome caractéristique de A, est **scindé à racines simples**. Donc A est diagonalisable dans $\mathbb{R}$.

Pour avoir les valeurs propres triées par ordre croissant on utilise la procédure suivante trouvée sur StackOverflow:

1. $idx = vp.argsort()[:: 1]$, calcul les indices de tri pour le vecteur vp et les met dans un vecteur idx.
2. $vp = vp[idx]$, tri le vecteur vp en utilisant les indices idx.
3. $vep = vep[: https://www.overleaf.com/download/project/64477dcc94aacf5aa6ad96d1/build/187ccfb1$
   $6d89756daa38fbb8/output/output.pdf?compileGroup = standardclsiserverid =$
   $clsi-pre-emp-e2-f-szlfenable_pdf_caching = truepopupDownload =$
   $true, idx]$, applique la même transformation au vecteur vep.

On a donc trié les valeurs propres et leurs vecteurs propres respectif.
On met ensuite les valeurs propres sur la diagonale principale de D et les vecteurs propres dans une matrice de passage P.

On calcul l'inverse de P, $Pinv$, puis on test que le produit matriciel $PDPinv$ soit bien égal à notre matrice A de départ.

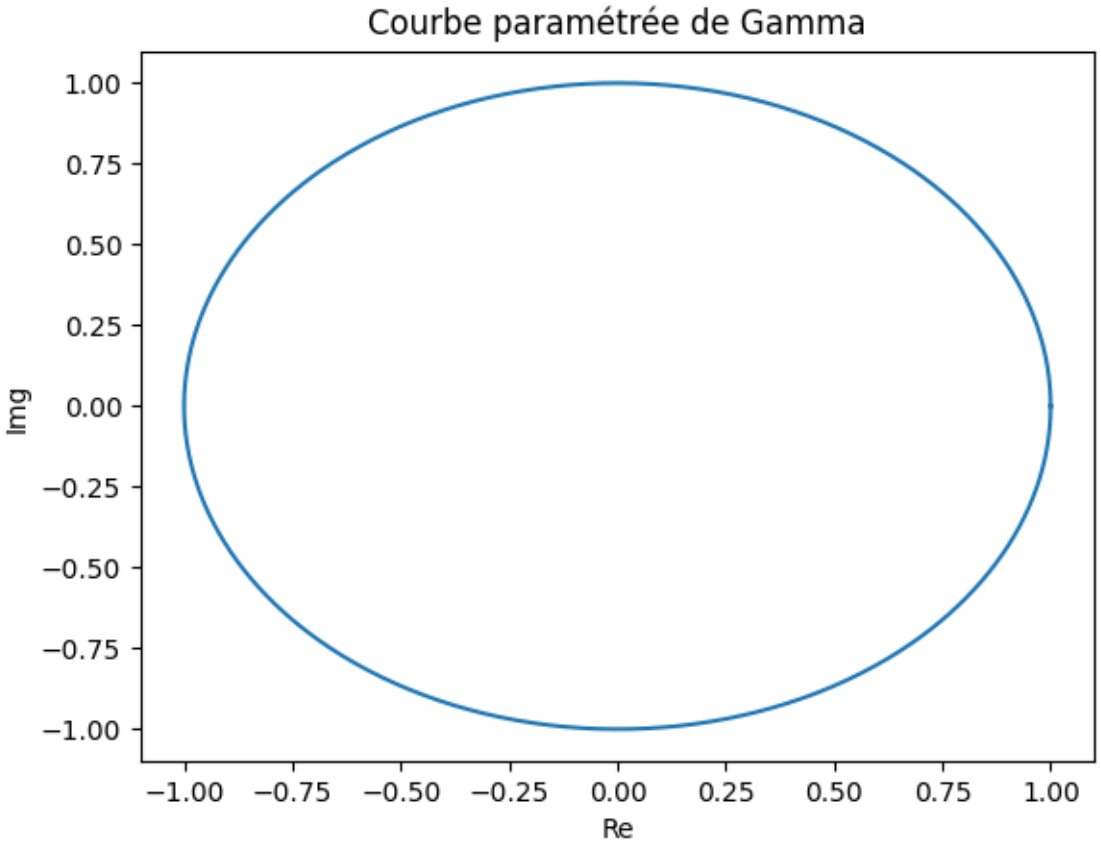
Remarque : e_j est un vecteur propre associé à la valeur propre de la j-eme colonne de D.

2 Recherche de valeur propre

On crée les fonctions *power_iteration* et *inverse_power_iteration* qui utilisent l'algorithme de la puissance itérée vu en cours. Ces foncions on d'abord été généréss par ChatGPT puis légèrement modifiées et testé.
On part d'un vecteur "aléatoire" et on itère en utilsant l'algorithme de la puissance itéré jusqu'à atteindre la précision voulu ou que le nombre d'itérations maximale soit atteind.
On aurait aussi pu utiliser une methode avec la décomposition QR.

3 Graphe d'une fonction paramétrée

On commence par crée notre fonction *gamma* à valeurs dans $\mathbb{C}$ à l'aide de l'unité complexe i codé par j en python.
On évalue ensuite *gamma* sur une subdivison régulière de l'intervalle [0,1] créer à l'aide de la commande *linspace*.
On extrait ensuite la partie réelle et la partie imaginaire puis on trace la courbe paramétrée avec en abscisse les parties réelles et en ordonné les parties imaginaires.
On remarque que *gamma* est un *lacet*.



4 Intégrale curviligne

Soit $f : \mathbb{R} \to \mathbb{C}$ une fonction à valeurs complexes d'une variable réelle t. Les parties réelles et imaginaires de f sont souvent dénotées $u(t)$ et $v(t)$, respectivement, de sorte que

$$f(t) = u(t) + iv(t). \tag{1}$$

Alors l'intégrale de f sur l'intervalle $[a, b]$ est donnée par

$$\int_a^b f(t)dt = \int_a^b (u(t) + iv(t))dt \ = \int_a^b u(t)dt + i\int_a^b v(t)dt. \tag{2}$$

Soit $f : \mathbb{C} \to \mathbb{C}$ une fonction continue sur une courbe lisse orientée γ. Soit $z : \mathbb{R} \to \mathbb{C}$ une paramétrisation de γ consistante avec sa direction. Alors l'intégrale le long de γ est notée

$$\int_\gamma f(z)dz \tag{3}$$

et se calcule par

$$\int_\gamma f(z)dz = \int_a^b f(\gamma(t))\gamma'(t)dt. \tag{4}$$

(`https://fr.wikipedia.org/wiki/M%C3%A9thodes_de_calcul_d%27int%C3%A9grales_de_contour`)

Pour appliquer la formule de l'intégrale curviligne d'une fonction le long d'un chemin on va avoir besoin de la dérivée du dis chemin. On commence donc par codé la fonction *dgamma*, la dérivée de *gamma*.
On crée notre fonction *integrale_curviligne* en appliquant la formule d'une intégrale curviligne.
Dans cette fonction on crée la fonction *intégrande* qui est la composée

$$f(\gamma(t))\gamma'(t)$$

On décompose la fonction integrande en parties réelle et imaginaire, et intègre chacune d'entre elles séparément en utilisant la fonction *quad* de la librairie *Scipy*. Finalement, on combine les deux intégrales pour obtenir l'intégrale curviligne complexe et on calcule l'erreur comme la somme des erreurs des intégrales réelles et imaginaires.
On test sur les 2 fonctions données en exemples.

5 Intégration numérique

On crée une fonction *rectangle_gauche* qui utilise la méthode des rectangles à gauches pour calculer numériquement l'intégrale d'une fonction donnée.
On prend bien soins d'initailiser notre résultat avec une valeur complexe.
On test d'abord notre fonction avec la fonction exponnentielle puis en faisant des intégrales curviligne (on donne les composés $f(\gamma(t))\gamma'(t)$ en argument à rectangle_gauche)

Remarque : La méthode est d'ordre > 1 car d'apres la question 3, gamma est une fonction régulière définie sur l'intervalle [0,1] et d'apres le papier cité en source du sujet (`https://irma.math.unistra.fr/~helluy/PREPRINTS/cras1998.pdf`) on peut utiliser la méthode dite de "périodisation".

6 Calcul symbolique

On utilise la librairie *sympy* pour définir un symbol t puis les fonctions *gamma*, *dgamma* (la dérivée de gamma), la matrice A de la question (1), puis notre intégrande f de l'intégrale curviligne

$$f = (\gamma(t)I_n - A)^{-1}\gamma'(t) \tag{5}$$

enfin on intègre sur l'intervalle [0,1].

Pour la somme des projections, comme γ entoure uniquement la valeur propre λ_1 on a,

$$\sum_{k\in\{1\}} \Pi_k = \Pi_1 = PJ_1P^{-1}$$

avec,

$$J_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Pour obtenir P on diagonalise A avec *sympy* et l'option $sort = True$ pour avoir les valeurs propres dans le même ordre qu'à la quesion 1.
On test en effectuant le produit matriciel PDP^{-1}.

7 Intégrale de contour numérique

On crée la fonction *contour* qui va calculer numériquement l'intégrale curviligne,

$$\frac{1}{2i\pi}\oint_\gamma (zI - A)^{-1}\mathrm{d}z$$

Le resultat est une matrice à valeurs complexes, on initialise donc dans notre fonction le résultat res avec une matrice nulle qu'on prend soins de sprécifier complexe qu'on va ensuite remplire.
On définie une fonction intégrande qui est la composée (5).
On intègre ensuite chaque entrée de la matrice intégrande en utilisant notre fonction *rectangle_gauche* définie à la question 5 qui prend bien des fonctions à valeurs complexes et on place le résultat dans res.

Pour la somme des projections,

$$\sum_{k\in K} \Pi_k = \sum_{k\in K} PJ_kP^{-1}$$

on crée une fonction *projection* qui prend comme arguments une matrice et un vecteur d'indices K.
Dans la fonction on commence par définir une projection individuel Π_i en utilisant la définition d'une projection et la fonction *diagonalize* définie à la question 1 pour récupérer P.
Il ne reste plus qu'à faire la somme des projections Π_i pour $i \in K$ et retourner le résultat.

8 Annexe

cf fichier *Annexe.md*