

# 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 efficacement 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 renvoie un couple de vecteurs (valeurs propre, vecteur propre). Comme les valeurs propre sont distincts,  $A$  est **scindé à racines simples** et donc 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.  $vep = vep[:,idx]$ , 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  $P$ ,  $P_{inv}$ , puis on test que le produit matriciel  $PDP_{inv}$  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 fonction ont d'abord été générés par ChatGPT puis légèrement modifié et testé.

On part d'un vecteur "aléatoire" et on itère en utilisant l'algorithme de la puissance itérée jusqu'à atteindre la précision voulu ou que le nombre d'itération maximale soit atteind.

On aurait aussi pu utiliser une methode avec la décomposition QR.

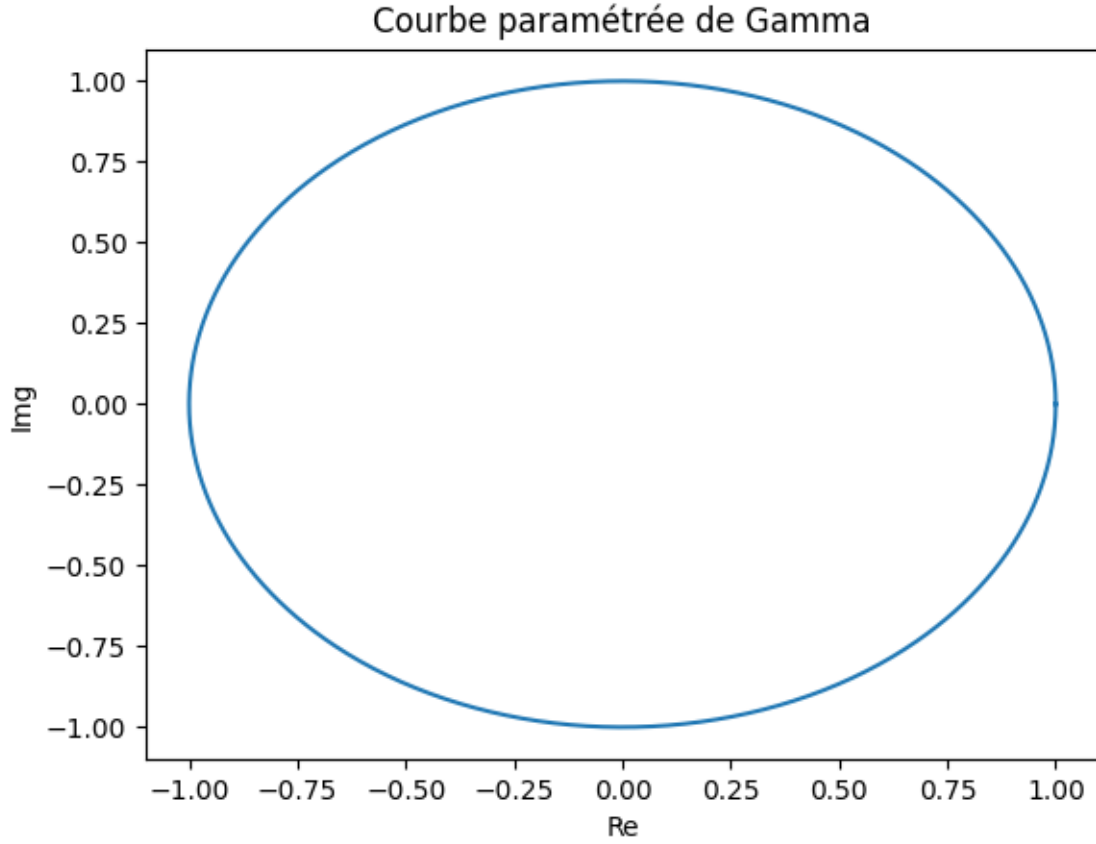
## 3 Graphe d'une fonction paramétrée

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  $f : \mathbb{R} \rightarrow \mathbb{C}$  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

$$f(t) = u(t) + iv(t). \quad (1)$$

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

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

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

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

et se calcule par

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

([https://fr.wikipedia.org/wiki/M%C3%A9thodes\\_de\\_calcul\\_d'int%C3%A9grales\\_de\\_contour](https://fr.wikipedia.org/wiki/M%C3%A9thodes_de_calcul_d'int%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 chemin. On commence donc par coder 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 *integrande* 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'initialiser notre résultat avec une valeur complexe.

On test d'abord notre fonction avec la fonction exponentielle 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'après la question (3), *gamma* est une fonction régulière définie sur l'intervalle  $[0,1]$  et d'après 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 de l'intégrale curviligne

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

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

Pour la somme des projections, comme  $\gamma$  entoure uniquement la valeur propre  $\lambda_1$  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 question (1). On teste en effectuant le produit matriciel  $PDP^{-1}$ .

## 7 Intégrale de contour numérique

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

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

Le résultat est une matrice à valeur complexe, on initialise donc dans notre fonction le résultat *res* avec une matrice nulle qu'on prend soin de préciser complexe qu'on va ensuite remplir.

On définit 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  $\Pi_i$  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  $\Pi_i$  pour  $i \in K$  et retourner le résultat.

## 8 Annexe

Person:

- Écrire un programme Python utilisant numpy qui permet de retrouver la plus grande et la plus petite valeur propre d'une matrice A au moyen de la méthode la puissance itérée, ainsi que des vecteurs propres associés

ChatGPT:

- Voici un exemple de programme Python qui utilise numpy pour calculer la plus grande et la plus petite valeur propre d'une matrice A en utilisant la méthode de la puissance itérée :

```
"""python

import numpy as np

def power_iteration(A, max_iter = 1000, tol = 1e - 8) :
"""
Calcule la plus grande valeur propre de la matrice A ainsi que le vecteur propre associé en utilisant la méthode de la puissance itérée.

Parameters
-----
A : numpy.ndarray
La matrice d'entrée de taille (n, n)
max_iter : int, optional
Le nombre maximum d'itérations de la méthode de la puissance itérée.
Default is 1000.
tol : float, optional
La tolérance pour la convergence de la méthode de la puissance itérée.
Default is 1e-8.

Returns
-----
(lam, v) : tuple of (float, numpy.ndarray)
La plus grande valeur propre de A et le vecteur propre associé.
"""
    n = A.shape[0]
    v = np.random.rand(n)
    lam = 0.0

    for i in range(max_iter):
        v_new = A @ v
        lam_new = np.linalg.norm(v_new)
        v_new = v_new / lam_new

        if np.abs(lam_new - lam) < tol:
```

```

break

    v = v_new
    lam = lam_new

    return (lam, v)

def inverse_power_iteration(A, max_iter=1000, tol=1e-8):
    """
    Calcule la plus petite valeur propre de la matrice A
    ainsi que le vecteur propre associé en utilisant la méthode
    de la puissance itérée inverse.

    Parameters
    -----
    A : numpy.ndarray
    La matrice d'entrée de taille (n, n)
    max_iter : int, optional
    Le nombre maximum d'itérations de la méthode de la puissance itérée inverse.
    Default is 1000.
    tol : float, optional
    La tolérance pour la convergence de la méthode de la puissance itérée inverse.
    Default is 1e-8.

    Returns
    -----
    (lam, v) : tuple of (float, numpy.ndarray)
    La plus petite valeur propre de A et le vecteur propre associé.
    """

    n = A.shape[0]
    v = np.random.rand(n)
    lam = 0.0

    for i in range(max_iter):
        v_new = np.linalg.solve(A, v)
        lam_new = np.linalg.norm(v_new)
        v_new = v_new/lam_new

        if np.abs(lam_new - lam) < tol:
            break

    v = v_new
    lam = lam_new

```

```
return (1/lam, v)
```

Exemple d'utilisation

```
A = np.array([[4, 2], [2, 3]])
```

Plus grande valeur propre et vecteur propre associé

```
lam1, v1 = power_iteration(A)
print("Plus grande valeur propre :", lam1)
print("Vecteur propre associé :", v1)
```

Plus petite valeur propre et vecteur propre associé

```
lam2, v2 = inverse_power_iteration(A)
print("Plus petite valeur propre :", lam2)
print("Vecteur propre associé :", v2)
"""
```

Dans cet exemple, la matrice d'entrée 'A' est définie comme une matrice '2x

[https://fr.wikipedia.org/wiki/M%C3%A9thodes\\_de\\_calcul\\_int%C3%A9grales\\_de\\_contour](https://fr.wikipedia.org/wiki/M%C3%A9thodes_de_calcul_int%C3%A9grales_de_contour)

- Person:

corrige cette fonction :

```
def integrale_curviligne(f, gamma, dgamma, a=0, b=1):
    """
```

L'intégrale curviligne de f le long de gamma.

Parameters:

——— f : fonction.

gamma : fonction. Chemin de classe C1 par morceaux.

Dgamma : fonction.

Dérivée de gamma.

a,b : float.

Borne d'integrations, domaine de gamma.

Returns:

———

(res, err) : couple of (complexe, float).

Valeur de l'intégrale de f le long de gamma et l'erreur d'integration.

"""

```
Re = np.real(f(gamma(t)) * dgamma(t))
```

```
Im = np.imag(f(gamma(t)) * dgamma(t))
```

```
res1, e1 = sc.integrate.quad(lambda t: Re, a, b)
```

```
res2, e2 = sc.integrate.quad(lambda t: Im, a, b)
```

```
return res1 + 1j*res2, e1+e2
```



- ChatGpt:

Il y a plusieurs erreurs dans cette fonction. Voici une version corrigée :

```
"""python
import numpy as np
import scipy.integrate as spi

def integralecurviligne(f, gamma, dgamma, a = 0, b = 1) :
    """
    L'intégralecurvilignede flelongdegamma.

    Parameters:
    -----
    f : fonction.
    gamma : fonction.
    Chemin de classe C1 par morceaux.
    dgamma : fonction.
    Dérivée de gamma.
    a,b : float.
    Borne d'intégrations, domaine de gamma.

    Returns:
    -----
    (res, err) : couple of (complexe, float).
    Valeur de l'intégrale de f le long de gamma et l'erreur d'integration.
    """
    def integrande(t):
        return f(gamma(t)) * dgamma(t)

    Intégration de la partie réelle de l'intégrale curviligne
    Re, Re_err = spi.quad(lambdat : np.real(integrande(t)), a, b)

    Intégration de la partie imaginaire de l'intégrale curviligne
    Im, Im_err = spi.quad(lambdat : np.imag(integrande(t)), a, b)

    Calcul de l'intégrale curviligne complexe et de l'erreur
    res = Re + 1j * Im
    err = np.sqrt(Re_err**2 + Im_err**2)

    return res, err
```

Voici les corrections que j'ai apportées :

- J'ai ajouté l'importation de la bibliothèque 'scipy.integrate' sous le nom 'spi', pour éviter de devoir taper 'scipy.integrate' à chaque fois.
- J'ai corrigé l'orthographe de 'dgamma' dans la liste des paramètres.

- J'ai défini une fonction 'integrande' qui calcule l'intégrande 'f(gamma(t)) \* dgamma(t)'.
- J'ai utilisé la fonction 'quad' de la bibliothèque 'scipy.integrate' pour calculer l'intégrale curviligne de la partie réelle et de la partie imaginaire de 'integrande'.
- J'ai calculé l'intégrale curviligne complexe en combinant les résultats de l'intégration de la partie réelle et de la partie imaginaire, et j'ai calculé l'erreur en combinant les erreurs de l'intégration de la partie réelle et de la partie imaginaire.

- Person:  
diagonalize [[3, 3, 4], [6, -2, -12], [-2, 3, 9]] using sympy

- ChatGPT:  
Voici comment diagonaliser la matrice [[3, 3, 4], [6, -2, -12], [-2, 3, 9]] en utilisant SymPy :

```
python

import sympy as sp

Définir la matrice A
A = sp.Matrix([[3, 3, 4], [6, -2, -12], [-2, 3, 9]])

Obtenir les vecteurs propres et les valeurs propres de A
eigenvectors = A.eigenvecs()

Diagonaliser la matrice A
D, P = A.diagonalize()

Afficher les vecteurs propres et les valeurs propres
print("Vecteurs propres et valeurs propres :")
for eigenvalue, multiplicity, eigenvects in eigenvectors:
    print("Valeur propre :", eigenvalue)
    for eigenvect in eigenvects:
        print("Vecteur propre :", eigenvect)
    print()

Afficher la matrice diagonalisée de A
print("Matrice diagonalisée de A :")
print(P**-1 * A * P)
```

La sortie sera :

Vecteurs propres et valeurs propres :  
Valeur propre : -4  
Vecteur propre : Matrix([[-1], [2], [1]])

Valeur propre : 6  
Vecteur propre :  $\text{Matrix}([2, 0, 1])$

Valeur propre : 8  
Vecteur propre :  $\text{Matrix}([1, -1, 1])$

Matrice diagonalisée de A :  
 $\text{Matrix}([$   
0, 0  
, 6, 0  
, 0, 8  
)

<http://galusins.univ-tln.fr/ENSEIGN/M43-sympy.html>

<https://docs.sympy.org/latest/modules/matrices/matrices.html>