TP 3 : Quelques méthodes numériques

I. Méthode de la puissance

Dans la suite, A désigne une matrice de $\mathcal{M}_n(\mathbb{R})$ diagonalisable. On pose $\lambda_{\max}$, la valeur propre la plus grande en valeur absolue de la matrice A.
Le but de cet exercice est de déterminer une approximation d'un vecteur propre de A associé à la valeur propre $\lambda_{\max}$. Pour cela, on définit la suite de vecteurs $(\mathrm{X}_k)_{k\in\mathbb{N}}$ par

$$\mathrm{X}_0 \in \mathcal{M}_{n,1}(\mathbb{R}) \quad \text{et} \quad \forall\, k \in \mathbb{N}, \quad \mathrm{X}_{k+1} = \frac{\mathrm{AX}_k}{\|\mathrm{AX}_k\|}$$

où $\|\cdot\|$ désigne la norme associée au produit scalaire canonique sur $\mathcal{M}_{n,1}(\mathbb{R})$. On admet que la suite $(\mathrm{X}_k)_{k\in\mathbb{N}}$ converge vers un vecteur propre associé à $\lambda_{\max}$.

1. Écrire une fonction **norme** qui prend en argument une matrice colonne X et renvoie sa norme $\|\mathrm{X}\|$.
2. Écrire une fonction **puissance(A, X0, k)** qui renvoie le terme X_k.
3. *Test.* On pose
$$\mathrm{A} = \begin{bmatrix} 0.5 & 0.5 \\ 0.2 & 0.8 \end{bmatrix} \quad \text{et} \quad \mathrm{X}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$
Calculer (sans Python) les valeurs propres de A et donner un vecteur propre de norme 1 associé à la plus grande valeur propre. Comparer les résultats obtenus avec les résultats obtenus avec la fonction **puissance**.
4. Modifier le programme pour afficher la plus petite valeur k telle que $\|\mathrm{AX}_k - \lambda_{\max}\mathrm{X}_k\| \leqslant 10^{-6}$ où ρ est la plus grande valeur propre.
5. On suppose la matrice A inversible. Comment obtenir une approximation d'un vecteur propre d'une matrice associé à la valeur propre la plus *petite* (en valeur absolue) ?

II. Méthode de Newton pour la recherche de zéro

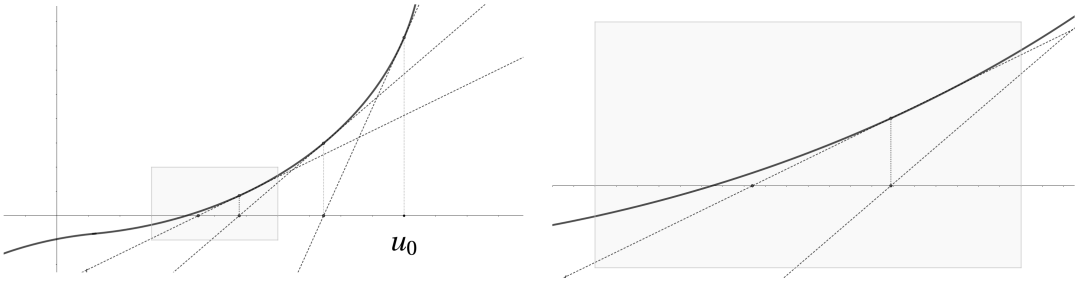
Rappel : en dimension 1

Partie A - Étude théorique

Soit $f : [a;b] \to \mathbb{R}$ de classe $\mathcal{C}^1$. On suppose que f' ne s'annule pas et que f s'annule une unique fois en un point c. On construit la suite u de la façon suivante :

- On part de $u_0 = a$.
- Pour tout $n \in \mathbb{N}$, u_{n+1} est l'abscisse du point d'intersection de l'axe des abscisses et de la tangente à la courbe représentative de f au point d'abscisse u_n.

1. Ci-dessous la courbe de f et des tangentes. Placer u_1, u_2 et u_3.
Conjecturer le comportement limite de la suite $(u_n)_{n\in\mathbb{N}}$.



2. Écrire l'équation de la tangente au point d'abscisse u_n. Vérifier que pour tout $n \in \mathbb{N}$,
$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}.$$
3. Si la suite u converge vers une limite finie, justifier que u converge vers c.
4. Dans cette question, on prend l'exemple de $f : x \in [1;2] \mapsto x^2 - 2$.
   (a) Vérifier que pour tout $n \in \mathbb{N}$, $\quad u_{n+1} - \sqrt{2} = \dfrac{\left(u_n - \sqrt{2}\right)^2}{2u_n}$.
   Puis, $\quad \left|u_{n+1} - \sqrt{2}\right| \leqslant \left|u_n - \sqrt{2}\right|^2 \quad$ et $\quad \left|u_n - \sqrt{2}\right| \leqslant |u_0 - \sqrt{2}|^{2^n}$.
   (b) En remarquant que $\left|u_0 - \sqrt{2}\right| \leqslant 1/2$, démontrer la convergence de la suite u vers $\sqrt{2}$.

Partie B - Python

5. Écrire un programme qui prend en entrée un entier n, u_0, f et f' et renvoie u_n.
6. Déduire de la question 4. une fonction Python qui prend en argument une précision p et renvoie une approximation de $\sqrt{2}$ à p-près.

En dimension n

Soit F une fonction de $\mathbb{R}^n$ à valeurs dans $\mathbb{R}^n$. Une telle fonction est définie par ses m fonctions composantes à valeurs réelles
$$\mathrm{F} : \boldsymbol{x} \in \mathbb{R}^n \mapsto \big(\ f_1(\boldsymbol{x}),\quad f_2(\boldsymbol{x}),\quad \ldots,\quad f_n(\boldsymbol{x})\ \big) \in \mathbb{R}^n.$$
On suppose dans la suite que toutes les fonctions composantes f_i sont de classe $\mathcal{C}^1$ sur $\mathbb{R}^n$ et on définit la matrice jacobienne de F par :
$$\mathrm{J}(\boldsymbol{x}) = \begin{bmatrix} \partial_1 f_1(\boldsymbol{x}) & \partial_2 f_1(\boldsymbol{x}) & \cdots & \partial_n f_1(\boldsymbol{x}) \\ \partial_1 f_2(\boldsymbol{x}) & \partial_2 f_2(\boldsymbol{x}) & \cdots & \partial_n f_2(\boldsymbol{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \partial_1 f_n(\boldsymbol{x}) & \partial_2 f_n(\boldsymbol{x}) & \cdots & \partial_n f_n(\boldsymbol{x}) \end{bmatrix}.$$
La méthode de Newton se généralise à plusieurs dimensions pour résoudre l'équation
$$\mathrm{F}(\boldsymbol{x}) = 0_{\mathbb{R}^n} \quad \text{d'inconnue } \boldsymbol{x} \in \mathbb{R}^n \qquad (\bullet)$$
Lorsque la matrice Jacobienne est inversible, on définit la suite $(\boldsymbol{x}_i)_{i\in\mathbb{N}}$ par $\boldsymbol{x}_0$ et
$$\forall\, i \in \mathbb{N}, \quad \boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \mathrm{J}(\boldsymbol{x}_i)^{-1}\mathrm{F}\left(\boldsymbol{x}_i\right)$$
où on identifie $\mathcal{M}_{n,1}(\mathbb{R})$ avec $\mathbb{R}^n$. Sous certaines conditions, on montre que la suite $(\boldsymbol{x}_i)_{i\in\mathbb{N}}$ converge vers une solution de l'équation $(\bullet)$.

Exemple
Considérons le système
$$\begin{cases} \cos(x) &=& \sin(y) \\ \mathrm{e}^{-x} &=& \cos(y) \end{cases} \quad \text{d'inconnue } (x,y) \in \mathbb{R}^2 \qquad (\star)$$

1. Déterminer une fonction F : $\mathbb{R}^2 \to \mathbb{R}^2$ de sorte que l'équation $(\star)$ soit équivalente à $\mathrm{F}(x,y) = (0,0)$. Préciser la matrice jacobienne de F. En déduire deux programmes, un pour le calcul de $\mathrm{F}(x,y)$, un autre pour $\mathrm{J}(x,y)$.
2. En déduire un programme et une approximation d'une solution de $(\star)$.

La méthode de Newton est assez efficace et permet une convergence rapide vers une solution. Cette méthode a de nombreuses variantes et extensions. Citons par exemple, son application dans les « théorèmes K.A.M » dont l'étude de la stabilité du système solaire.

Solution

I.1.

```
import numpy as np

def norme(X):
    n=len(X)
    s=0
    for i in range(n):
        s=s+X[i]**2
    return float(s**(1/2))
```

I. 2.

```
def puissance(A,X0,k):

    X=X0
    for i in range(k):
        X=np.dot(A,X)
        X=X/norme(X)
    return X
```

I. 3. On vérifie que le spectre est constitué de 0.3 et 1. De plus, un vecteur propre associé à $\lambda_{\max} = 1$ est donné par

$$\mathrm{X}_p = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Pour avoir un vecteur propre de norme 1, il suffit de diviser le vecteur précédent par sa norme : $\sqrt{2}$.
Testons maintenant le programme :

```
Atest=np.array([[0.5,0.5],[0.2,0.8]])
X0test=np.array([[1],[0]])
print(puissance(Atest,X0test,10))
```

```
[[0.70711409]
 [0.70709947]]
```

On retrouve bien un vecteur "presque" colinéaire au vecteur propre X_p.

I.4.

```
def seuil(A,X0):
    X=X0
    k=0
    err=norme(np.dot(A,X)-X)
    while err>10**(-6):
        X=np.dot(A,X)
        X=X/norme(X)
        err=norme(np.dot(A,X)-X)
        k=k+1
    return k

print(seuil(Atest,X0test))
```

On trouve 12.

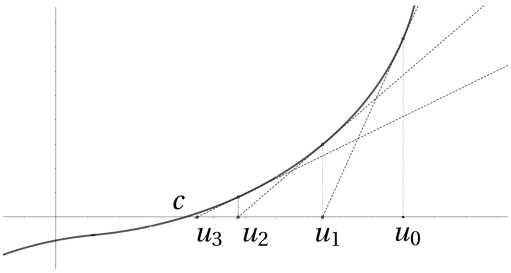
I.5 Si la matrice est inversible, λ est valeur propre de A si et seulement si λ^{-1} est valeur propre de A^{-1}. L'idée est donc d'appliquer la méthode de la puissance à l'inverse.

```
Ainv=np.linalg.inv(Atest)
print(puissance(Ainv,X0test,100))
```

```
[[ 0.92847669]
 [-0.37139068]]
```

II. Dimension 1

On conjecture que la suite u tend vers c, le zéro de la fonction f.



2. L'équation au point d'abscisse u_n est

$$y = f(u_n) + f'(u_n)(x - u_n).$$

On résout donc l'équation d'inconnue $x \in \mathbb{R}$,

$$0 = f(u_n) + f'(u_n)(x - u_n).$$

Comme $f'(u_n) \neq 0$, on trouve une unique solution donnée par :

$$\boxed{u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}.}$$

3. Supposons qu'il existe $\ell \in \mathbb{R}$ tel que $u_n \underset{n\to\infty}{\longrightarrow} \ell$.
Par continuité de f et f', on a aussi :

$$f(u_n) \underset{n\to\infty}{\longrightarrow} f(\ell) \quad \text{et} \quad f'(u_n) \underset{n\to\infty}{\longrightarrow} f'(\ell).$$

Comme f' ne s'annule pas, on a par somme et quotient :

$$u_n - \frac{f(u_n)}{f'(u_n)} \underset{n\to\infty}{\longrightarrow} \ell - \frac{f(\ell)}{f'(\ell)}.$$

De plus, on a par définition de u,

$$\forall n \in \mathbb{N}, \quad u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}.$$

Par unicité de la limite, $\ell = \ell - \dfrac{f(\ell)}{f'(\ell)}$.

Nécessairement, $\boxed{f(\ell) = 0.}$

Retenons que si la suite u converge, alors elle converge vers l'unique zéro de f.

4.(a) La fonction f est bien de classe $\mathcal{C}^1$ car polynomiale. Elle ne s'annule qu'en $\sqrt{2}$. De plus, pour tout $x \in [1;2]$, $f'(x) = 2x$. On en déduit que pour tout $n \in \mathbb{N}$,

$$u_{n+1} = u_n - \frac{{u_n}^2 - 2}{2u_n}.$$

Expression que l'on simplifie sous la forme :

$$u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right).$$

Dans ce cas, on obtient

$$\begin{aligned} u_{n+1} - \sqrt{2} &= \frac{1}{2}\left(u_n + \frac{2}{u_n}\right) - \sqrt{2} \\ &= \frac{{u_n}^2 - 2\sqrt{2}u_n + 2}{2u_n} \\ u_{n+1} - \sqrt{2} &= \frac{(u_n - \sqrt{2})^2}{2u_n}. \end{aligned}$$

Comme $2u_n \geqslant 1$, $1/(2u_n) \leqslant 1$ et

$$\left|u_{n+1} - \sqrt{2}\right| \leqslant \frac{\left|u_n - \sqrt{2}\right|^2}{2u_n} \leqslant \left|u_n - \sqrt{2}\right|^2.$$

- Procédons par récurrence sur la propriété :

$$\mathcal{P}(n) : \qquad \left|u_n - \sqrt{2}\right| \leqslant |u_0 - \sqrt{2}|^{2^n}.$$

Initialisation.
On a $|u_0 - \sqrt{2}| = |u_0 - \sqrt{2}|^{2^0}$.
La propriété $\mathcal{P}(0)$ est donc vraie.

Hérédité. Soit $n \in \mathbb{N}$. Supposons $\mathcal{P}(n)$ vraie et démontrons que $\mathcal{P}(n+1)$ est vraie. On a d'après ce qui précède

$$\begin{aligned} \left|u_{n+1} - \sqrt{2}\right| &\leqslant |u_n - \sqrt{2}|^2 \\ &\leqslant \left||u_0 - \sqrt{2}|^{2^n}\right|^2 \\ &\leqslant |u_0 - \sqrt{2}|^{2^n \cdot 2} \\ &\leqslant |u_0 - \sqrt{2}|^{2^{n+1}}. \end{aligned}$$

$\mathcal{P}(n+1)$ est donc bien vérifiée.

Conclusion. Pour tout entier n, $\mathcal{P}(n)$ est vraie.

4.(b) À l'aide de la remarque $|u_0 - \sqrt{2}| < 1$, puis,

$$\left|u_0 - \sqrt{2}\right|^{2^n} \underset{n\to\infty}{\longrightarrow} 0.$$

Par encadrement, $\boxed{u_n \underset{n\to\infty}{\longrightarrow} \sqrt{2}.}$

5. Un programme possible est :

```
def u_n(fonct,der,u0,n) :
    u = u0
    for i in range(n) :
        u = u-fonct(u)/der(u)
    return u
```

6. Donnons un code à l'aide d'une boucle while.

```
def approx(p) :
    # p est la précision
    u=1
    erreur=1/2
    while erreur>p :
        erreur=erreur**2
        u=(u+2/u)/2
    # La formule de récurrence de la
        suite u
    return u
```

La méthode de Newton est à comparer à la méthode d'approximation par dichotomie. La convergence de la suite obtenue par la méthode de Newton est bien plus rapide que celle par dichotomie. Il faut cependant supposer la fonction f dérivable avec que la dichotomie ne suppose que la continuité.

II. Dimension n

1.

```
import numpy as np

def F(X):
    a=float(np.cos(X[0])-np.sin(X[1]))
    b=float(np.exp(-X[0])-np.cos(X[1]))
    return np.array([[a],[b]])

def jacobienne(X):
    a=float(-np.sin(X[0]))
    b=float(-np.cos(X[1]))
    c=float(-np.exp(-X[0]))
    d=float(np.sin(X[1]))
    return np.array([[a,b],[c,d]])
```

2.

```
def approximation(n):
    X=np.array([[1],[0]])
    for i in range(n):
        Jinv=np.linalg.inv(jacobienne(X))
        X=X-np.dot(Jinv,F(X))
    return X
```

Une solution est approximativement donnée par $x = 0.58853$ et et $y = 0.98226$.