TP de Python numéro 6
Approximations numériques

*Semaine du **jeudi 5 décembre**.*

Le but de ce TP est d'étudier des méthodes générales permettant d'obtenir des approximations numériques de réels dans certains contextes :

- des approximations numériques de solutions d'équations sous la forme $f(x) = 0$, où f est une fonction *continue*, et
- des approximations numériques de limites de suites.

On verra également que ces deux contextes sont liés dans un certain nombre de méthodes.

Ces méthodes, dites de *calcul numérique*, permettent en particulier d'avoir des approximations de solutions qu'on ne sait pas résoudre *algébriquement*, c'est-à-dire pour lesquelles on ne sait pas trouver d'expression explicite.

Exemple 1. Il n'existe pas d'expression explicite (utilisant les fonctions usuelles) donnant la ou les solutions de l'équation $xe^x = y$ (de paramètre $y \in \mathbb{R}$ et d'inconnue réelle x) en fonction de y.

I. Approximations numériques de solutions d'équations de la forme $f(x) = 0$

Dans cette partie, on considère un intervalle I (contenant au moins deux points) et une fonction $f : I \to \mathbb{R}$ qui est *continue sur I*.

On va voir des méthodes fournissant des approximations, à une précision voulue, d'une solution de l'équation

$$f(x) = 0.$$

On se placera donc toujours dans un contexte où l'on sait que l'équation $f(x) = 0$ admet au moins une solution sur I.

L'idée principale sous-jacente à ces méthodes est le théorème des valeurs intermédiaires, sous la forme suivante.

Théorème 2. *Soient I un intervalle, $f : I \to \mathbb{R}$ une fonction continue sur I et a et b deux points de I tels que $a < b$. Si $f(a)$ et $f(b)$ ne sont pas de même signe, alors f s'annule sur le segment $[a, b]$.*

Remarque.

- D'après la règle des signes, une manière algébrique de signifier que $f(a)$ et $f(b)$ n'ont pas le même signe est donnée par :
$$f(a)f(b) \leq 0.$$
Nous utiliserons **systématiquement** cette idée dans nos algorithmes, il faut être bien capable d'effectuer cette traduction.
- Le théorème des valeurs intermédiaires a bien d'autres formes que nous verrons dans le chapitre sur la continuité, mais elles sont équivalentes à ce théorème.
- La 2e méthode que nous allons voir, utilisant le principe de dichotomie (voir TP précédent), n'est pas à proprement parler une utilisation du théorème intermédiaire mais bien plus : l'idée derrière cette méthode **fournit la démonstration de ce théorème** (voir chapitre sur la continuité).

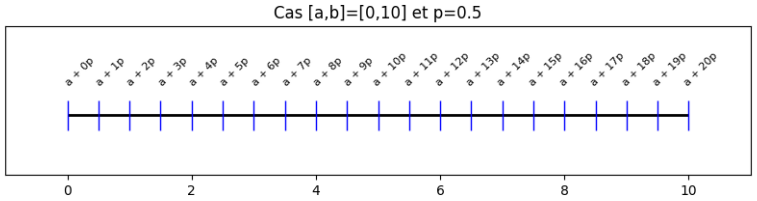
1. Recherche par balayage

Dans les paragraphes théoriques de cette partie, a et b sont deux réels tels que $a < b$, et f est une fonction continue sur $[a, b]$ telle que $f(a)f(b) < 0$.

La fonction f change de signe sur $[a, b]$ donc s'annule sur $[a, b]$ (par le TVI), et la méthode de recherche par balayage permet d'obtenir une approximation d'un réel x en lequel $f(x) = 0$.

Enfin, on fixe un réel $p > 0$ qui est la précision de l'approximation recherchée.

La recherche par balayage (considérée assez *naïve*, car simple et peu efficace) consiste à tester si f change de signe entre les points de la forme $a, a + p, a + 2p, a + 3p, ..., a + kp, ...$



Plus précisément, si l'on détecte un changement de signe de f entre les points $a + kp$ et $a + (k + 1)p$ pour un certain k, alors on sait que f s'annule sur le segment $[a + kp, a + (k + 1)p]$, de longueur p, et son milieu $a + (k + \frac{1}{2})p$ fournit alors une approximation avec précision p (et même en fait $\frac{p}{2}$) d'une solution de l'équation $f(x) = 0$.

Remarque. Ce changement de signe a lieu pour le plus petit $k \in \mathbb{N}$ tel que $f(a)f(a + (k + 1)p)$ devient négatif (car alors, $f(a)$ et $f(a + kp)$ ont le même signe, alors que $f(a)$ et $f(a + (k + 1)p)$ sont de signe opposés).

L'algorithme de recherche par balayage s'organise comme suit.

- On déclare une variable `k` initialisée à 0.
- Tant que $f(a)f(a + (\texttt{k} + 1)p) > 0$, on incrémente `k` de 1.
- Fin de la boucle précédente : on renvoie $a + (\texttt{k} + \frac{1}{2})p$, qui est l'approximation recherchée d'une solution de $f(x) = 0$.

Remarque. En fait, dans cette méthode, on utilise le point b uniquement pour savoir que f s'annule en un point supérieur à a (et donc que cette recherche aboutira).

Remarque. On peut toujours mettre une équation donnée sous la forme $f(x) = 0$, en faisant passer le membre de droite à gauche du signe égal par soustraction.

Exercice 3. On souhaite appliquer la méthode de recherche par balayage pour déterminer une approximation à 10^{-6} près d'une solution positive de l'équation $(E) : xe^x = 1$.

1. Quelle fonction f faut-il poser pour mettre (E) sous la forme $f(x) = 0$?
2. Justifier qu'une solution positive de (E) existe, et qu'on peut appliquer la méthode de recherche par balayage à f à partir de 0.
3. Écrire le code d'une fonction Python d'entête `def f(x):` prenant en entrée un réel x et renvoyant en sortie $f(x)$.
4. Utiliser l'algorithme de recherche par balayage décrit plus haut pour obtenir une approximation à 10^{-6} près d'une solution positive de l'équation (E).

Remarque. Plus on veut une approximation précise, plus le paramètre `p` choisi sera proche de 0 et plus l'algorithme effectuera de calculs.

Plutôt que de réécrire à chaque fois les étapes de l'algorithme de recherche par balayage, on peut l'implémenter une fois pour toute pour s'en resservir plus facilement.

Remarque. En Python, on peut tout à fait manipuler une fonction Python comme n'importe quelle variable, et la mettre en argument d'une autre fonction.

Exercice 4. Écrire le code d'une fonction d'entête `def balayage(f, a, p):` prenant en entrée :

- une fonction Python `f` qui implémente une fonction mathématique f,
- un réel `a` et un réel strictement positif `p`,

- ces données étant soumises aux hypothèses de cette partie (f est continue sur un intervalle contenant `a` et s'annule sur cet intervalle),

et renvoyant en sortie une approximation à précision `p` d'une solution supérieure à `a` de l'équation $f(x) = 0$.

Remarque. Rappel : pour tout réel positif y, $\sqrt{y}$ est défini comme l'unique réel positif x vérifiant $x^2 = y$.

Exercice 5. Utiliser l'exercice précédent et la remarque ci-dessus pour donner des approximations à 10^{-5} près des réels $\sqrt{10}$ et $\sqrt{28}$. Comparez votre résultat à celui donné par la fonctions `np.sqrt`.

Des fois, il peut être nécessaire de trouver le point de départ a vous-même avant d'utiliser la recherche par balayage.

Exercice 6. En utilisant la méthode de recherche par balayage, déterminer une approximation à 10^{-5} près d'une solution de l'équation $x^5 - 100x^3 + 20x + 1 = -20$.

2. La recherche par dichotomie

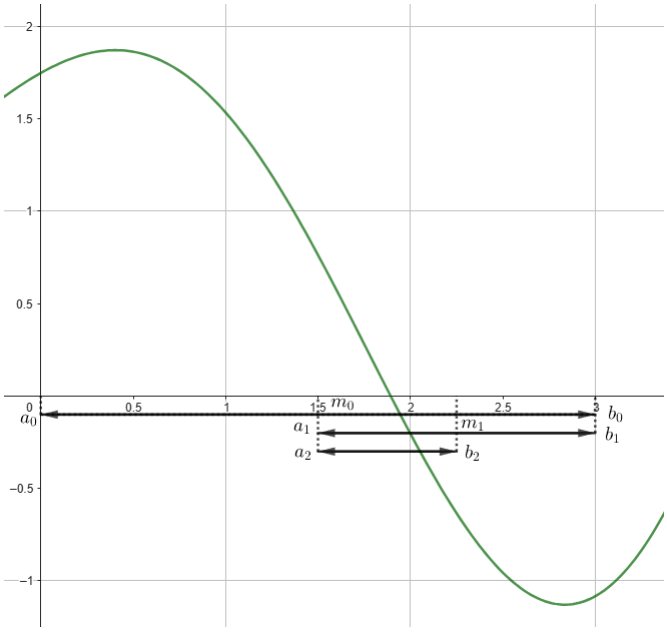
C'est une méthode plus riche qui, en plus de fournir un algorithme d'approximation, permet de démontrer le théorème des valeurs intermédiaires. On rappelle que le mot *dichotomie* est d'origine grecque et signifie *couper en deux*.

- On considère toujours une fonction f continue sur un segment $[a, b]$ fixé (avec a et b réels tels que $a < b$) et telle que $f(a)f(b) \leq 0$.
- On cherche une approximation à précision $p > 0$ fixée d'une solution sur $[a, b]$ de l'équation $f(x) = 0$.

L'idée est la suivante:

- L'algorithme s'effectue par étapes. À l'étape n, on va affiner notre segment de recherche $[a_n, b_n]$ d'une solution de $f(x) = 0$ en un segment $[a_{n+1}, b_{n+1}]$ de longueur moitié.
- Initialement, à l'étape $n = 0$, on sait que f s'annule sur $[a, b]$ et on pose donc $a_0 = a$ et $b_0 = b$.
- Pour passer de l'étape n à l'étape $n + 1$, on considère le milieu $m_n = \dfrac{a_n + b_n}{2}$ du segment $[a_n, b_n]$. On a donc $a_n < m_n < b_n$ et :
  - Si f change de signe entre a_n et m_n, on pose $a_{n+1} = a_n$ et $b_{n+1} = m_n$.
  - Sinon, f change de signe entre m_n et b_n, on pose alors $a_{n+1} = m_n$ et $b_{n+1} = b_n$.

  Dans tous les cas, on construit ainsi un nouvel intervalle $[a_{n+1}, b_{n+1}]$ de longueur moitié par rapport à $[a_n, b_n]$ sur lequel f change encore de signe donc s'annule.
- On arrête l'algorithme à la première étape n pour laquelle le segment de recherche $[a_n, b_n]$ devient de longueur inférieure à p, et on renvoie son milieu $\dfrac{a_n + b_n}{2}$ comme approximation voulue.



Exercice 7. On considère l'équation $(E) : e^{-x^2} = 2x$, d'inconnue réelle x.

1. Quelle fonction f faut-il considérer pour mettre (E) sous la forme $f(x) = 0$? Vérifier que f change de signe entre 0 et 2.
2. Compléter le code Python ci-dessous pour qu'il affiche une approximation à 10^{-6} près d'une solution de (E) sur $[0, 2]$, en utilisant la recherche par dichotomie.

```
import numpy as np

# définition de f
def f(x):
    return ...

# Intervalle de recherche et précision
a,b=0,2
p=10**(-6)

while  ...  : # exprimer la condition d'arrêt
    # milieu de la zone de recherche
    m=(a+b)/2
    # mise à jour de la zone de recherche
    if   ... ... :
        ... = m
    else :
        ... = m
# fin de boucle : on affiche l'approximation
print((a+b)/2)
```

Exercice 8. Écrire le code d'une fonction d'entête `def dichotomie(f, a, b, p):` prenant en entrée :

- Une fonction Python `f` qui implémente une fonction mathématique f continue,
- deux réels `a` et `b` entre lesquels f change de signe,
- un réel `p`> 0

et renvoyant en sortie une approximation à précision `p` d'une solution de l'équation $f(x) = 0$ sur $[\texttt{a}, \texttt{b}]$ obtenue par la méthode de recherche par dichotomie.

Remarque. En pratique, il pourrait être difficile de trouver les deux points a et b pour appliquer l'algorithme. On peut alors effectuer un balayage grossier pour déterminer deux points entre lesquels f change de signe.

Exercice 9. On considère l'équation $(E) : x^5 - 123x = 123$. On considère le polynôme $f : x \mapsto x^5 - 123x - 123$. f est continue en tant que polynôme et permet d'écrire (E) sous la forme $f(x) = 0$. On remarque que :

- $f(0) = -123 < 0$,
- $f(x) \xrightarrow[x\to+\infty]{} +\infty$

de sorte que f change nécessairement de signe sur $\mathbb{R}_+$ (c'est le résultat d'un exercice classique sur la continuité).

1. Effectuer un balayage grossier à partir de 0 pour déterminer un réel b en lequel $f(b) > 0$.
2. En déduire une approximation à 10^{-8} près d'une solution positive de (E), obtenue par dichotomie.

3. La méthode de Newton

Nous allons moins rentrer dans les détails de cette méthode afin de rester concis, mais la méthode de Newton est d'une importance capitale en mathématiques (elle se généralise à énormément de contextes).

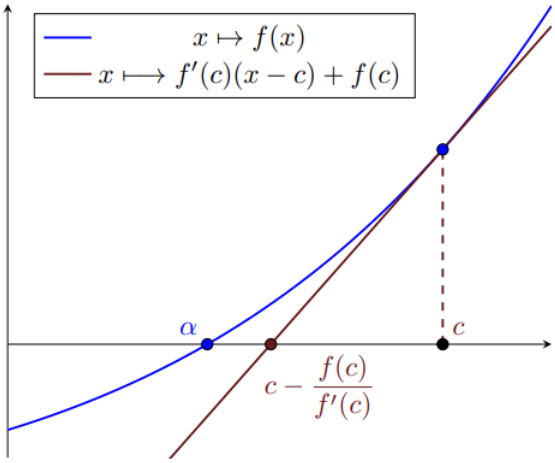
On souhaite toujours produire une approximation numérique d'une solution d'une équation de la forme (E) : $f(x) = 0$, où f est une fonction réelle.

On se placera dans le cadre suivant :

- On dispose d'un intervalle I sur lequel on sait que f s'annule.
- On suppose f dérivable sur l'intervalle I, et on suppose : $\forall x \in I, f'(x) \neq 0$.

On cherche alors une approximation d'une solution α de (E). L'idée est de partir d'un point c *suffisamment proche* de α (on restera évasif sur ce point), et d'utiliser l'approximation de f donnée par sa tangente pour se rapprocher de α.

Avec une étape, la méthode de Newton à partir de c fournit l'approximation $c - \dfrac{f(c)}{f'(c)}$ de α.



$$c - \frac{f(c)}{f'(c)} \text{ est l'unique solution de l'équation } f(c) + (x-c)f'(c) = 0.$$

L'idée de Newton est alors de répéter ce processus pour fournir une approximation de α.

Dans le cas où I est un segment $[a, b]$, on peut donc suivre l'algorithme suivant :

- Un considère un réel $u_0 \in [a, b]$,
- puis on calcule successivement des termes de la suite $(u_n)_{n\in\mathbb{N}}$ donnée par la relation de récurrence :

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

- On admettra sans le quantifier que pour N *suffisamment* grand, u_N fournit une bonne approximation d'une solution de l'équation $f(x) = 0$.

Cette méthode pose beaucoup de problèmes en pratique : les termes u_n calculés restent-ils dans l'intervalle I ? Combien de termes devons nous calculer pour avoir une approximation à 10^{-6} près ? Nous n'allons pas les résoudre ici.

En pratique, on se contentera de faire 10 à 100 itérations, en admettant que la méthode s'applique.

Exercice 10. On souhaite appliquer la méthode de Newton pour déterminer une approximation de la solution α sur $[0, 1]$ de l'équation $(E) : e^{-x} = x$.

1. Quelle fonction dérivable f doit-on poser pour mettre (E) sous la forme $f(x) = 0$? Écrire deux fonctions Python `f` et `f'` qui implémentent informatiquement les fonctions f et f'.
2. Justifier que (E) admet une unique solution α sur $[0, 1]$.
3. Utiliser la méthode de Newton pour donner une approximation de α. On procédera à 20 itérations de la méthode de Newton.
4. Comparer le résultat avec celui obtenu via la fonction `dichotomie` avec une précision de 10^{-8}.

II. Utilisations de suites et approximations

Un principe général pour cette partie : on peut utiliser des encadrements obtenus par des calculs théoriques afin de fournir des approximations numériques de nombres réels en Python.

1. Utilisation directe d'encadrements

Le contexte général aux exercices ci-dessous est le suivant :

- On souhaite donner une approximation à précision $p > 0$ donnée d'un réel α.
- On dispose pour cela d'une suite $(u_n)_n$ telle que $u_n \xrightarrow[n\to+\infty]{} \alpha$.
- On majore $|u_n - \alpha|$ par une suite explicite simple (en fonction de n) qui tend vers 0.
- On utilise cette majoration pour trouver un entier N pour lequel u_N fournit une approximation à précision p de α.

Ces méthodes s'utilisent particulièrement dans des méthodes dites *de point fixe*, qui permettent d'approximer des solutions d'équations de la forme $f(x) = 0$ (voir plus bas).

Commençons par un premier exemple simple.

Exemple 11. Soit $g : \left| \begin{array}{ccc} \mathbb{R}_+ & \longrightarrow & \mathbb{R} \\ x & \longmapsto & 1+\sqrt{x} \end{array} \right.$.
On considère la suite $u = (u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 2$ et $\forall n \in \mathbb{N}, u_{n+1} = g(u_n)$.

On suppose que, dans le cadre d'un exercice de mathématiques, on a démontré que $(u_n)_{n\in\mathbb{N}}$ admet une limite α, ainsi que bien d'autres résultats.

Puis, on tombe sur l'enchainement des questions suivantes.

- (N) Montrer que $\forall n \in \mathbb{N}, |u_n - \alpha| \leq \left(\frac{1}{2\sqrt{2}}\right)^n$.
- $(N+1)$ En déduire un code Python permettant d'afficher une approximation de α avec une précision de 10^{-6}.

L'idée, pour répondre à la question $(N+1)$, est d'utiliser l'encadrement de la question (N) :

$$\forall n \in \mathbb{N}, |u_n - \alpha| \leq \left(\frac{1}{2\sqrt{2}}\right)^n.$$

On écrit donc un code Python qui calcule successivement les termes de u jusqu'à avoir calculé un terme u_n pour lequel n est assez grand pour vérifier :

$$\left(\frac{1}{2\sqrt{2}}\right)^n \leq 10^{-6}.$$

Ceci est possible car $\left(\frac{1}{2\sqrt{2}}\right)^n \xrightarrow[n\to+\infty]{} 0$ (c'est une suite géométrique, de raison $\frac{1}{2\sqrt{2}} \in]-1,1[$).

Pour un tel entier n, on a alors $|u_n - \alpha| \leq 10^{-6}$ par transitivité.

Autrement dit, u_n fournit une approximation de α à 10^{-6} près. On écrit donc le code suivant.

```
import numpy as np
# définition de g
def g(x)=
    return(1+np.sqrt(x))

# Calculs de termes u(n) jusqu'à un n assez grand
u=2 # contient les termes successifs de u
n=0 # contient l'indice du terme présent dans u
q=1/(2*np.sqrt(2)) # simple raccourci
while q**n > 10**(-6): # condition d'arrêt avec le majorant obtenu
    u=g(u)
    n+=1
# fin de la boucle : n est assez grand
print(u)
```

Exercice 12. On fixe un entier $N \geq 1$ qui n'est pas le carré d'un entier. Le but de cet exercice est de fournir une approximation de $\sqrt{N}$ avec l'aide de Python, sans utiliser la fonction `np.sqrt` : c'est ce genre de démarche qui permet de définir la fonction informatique `np.sqrt`.

On considère pour cela la fonction $g : x \mapsto \dfrac{N+x^2}{2x}$, de domaine de définition sur $\mathbb{R}^*$.

1. Montrer qu'il existe un unique réel positif α tel que $g(\alpha) = \alpha$ et donner une expression de α en fonction de N.
   On dit que α est un point fixe de g.
2. Montrer que $[1, +\infty[$ est stable par g, c'est-à-dire que $g([1, +\infty[) \subset [1, +\infty[$.
3. Montrer que : $\forall x \in \mathbb{R}_+^*, g(x) - \alpha = \dfrac{(x-\alpha)^2}{2x}$.

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $\begin{cases} u_0 = \lfloor \alpha \rfloor \\ \forall n \in \mathbb{N}, u_{n+1} = g(u_n) \end{cases}$.

4. Montrer que pour tout $n \in \mathbb{N}$, $u_n \in [1, +\infty[$ et $|u_{n+1} - \alpha| \leq \dfrac{|u_n - \alpha|^2}{2}$.
5. En déduire que pour tout entier n, $|u_n - \alpha| \leq \left(\dfrac{1}{2}\right)^{2^n - 1}$, puis que $u_n \xrightarrow[n\to+\infty]{} \alpha$.
6. Écrire le code d'une fonction d'entête `def terme_initial(N):` prenant en entrée l'entier N et renvoyant $\lfloor \alpha \rfloor$ (sans utiliser `np.sqrt`).
7. En déduire le code d'une fonction Python `approx_racine` prenant en entrée l'entier N et un réel $p > 0$, et renvoyant en sortie une approximation à précision p de $\sqrt{N}$ obtenue sans utiliser la fonction `np.sqrt`.

2. Remarque générale sur les méthodes dites de points fixes

Soit f une fonction réelle, notons D son domaine de définition. Reprenons le contexte de la première partie, où l'on cherche à produire des approximations de solutions de l'équation $f(x) = 0$.

Définition 13. Soit g une fonction réelle. On appelle *point fixe de g* tout réel x du domaine de définition de g vérifiant :
$$g(x) = x.$$

Soit g la fonction définie sur D par $g(x) = f(x) + x$. Alors, pour tout réel $x \in D$, on a l'équivalence :
$$f(x) = 0 \iff g(x) = x.$$

Autrement dit, les solutions de l'équation $f(x) = 0$ sont exactement les points fixes de g.

D'autre part, dans un grand nombre de cas (quand g -ou de manière équivalente, f- est continue sur D), on peut montrer que si une suite $u \in D^{\mathbb{N}}$ vérifie : $\forall n \in \mathbb{N}, u_{n+1} = g(u_n)$, et admet une limite finie $l \in D$, alors l est un point fixe de g.

Démonstration. On suppose que $u_n \xrightarrow[n\to+\infty]{} l$, que $u_n \in D$ pour tout entier n et que $l \in D$. Par continuité de g en l, on a donc :
$$g(u_n) \xrightarrow[n\to+\infty]{} g(l).$$

Mais : $\forall n \in \mathbb{N}, u_{n+1} = g(u_n)$. Donc $u_{n+1} \xrightarrow[n\to+\infty]{} g(l)$.

Or, on sait que $u_n \xrightarrow[n\to+\infty]{} l$, donc $u_{n+1} \xrightarrow[n\to+\infty]{} l$.

Par unicité de la limite, on a donc $g(l) = l$: l est un point fixe de g. □

Cette remarque est souvent utilisée dans les exercices : pour déterminer une approximation d'une solution de $f(x) = 0$, on pose $g(x) = f(x) + x$ et une s'intéresse, conformément à la partie précédente, à une suite vérifiant la relation de récurrence $u_{n+1} = g(u_n)$.

Exemple 14. L'exercice 12 est un exemple ! $\sqrt{N}$ est l'unique solution positive de l'équation $f(x) = 0$, où $f : x \mapsto \dfrac{N - x^2}{2x}$ (la division par $2x$ rend la méthode plus efficace), et la fonction g n'est autre que $x \mapsto f(x) + x$.

3. Utilisation de suites adjacentes

On rappelle le théorème des suites adjacentes:

Théorème 15. *Soient $u = (u_n)_{n\in\mathbb{N}}$ et $v = (v_n)_{n\in\mathbb{N}}$ deux suites réelles. On suppose que :*

- *u est croissante, v est décroissante, et*
- *$v_n - u_n \xrightarrow[n\to+\infty]{} 0$.*

Alors, u et v convergent vers une limite commune l et :

$$\forall n \in \mathbb{N}, u_n \leq l \leq v_n.$$

En présence de telles suites adjacentes, et avec les notations de l'énoncé, on peut utiliser l'encadrement

$$\forall n \in \mathbb{N}, u_n \leq l \leq v_n$$

pour fournir des approximations de l.

En effet, pour avoir une approximation de l avec une précision $p > 0$, il suffit de calculer les termes de u_n et v_n jusqu'à trouver un entier n tel que $|v_n - u_n| < p$, car alors l'intervalle $[u_n, v_n]$ est de longueur au plus p et contient l. Pour un tel entier n, une approximation de l à précision p est alors donnée au choix par u_n, v_n ou par exemple $\dfrac{u_n + v_n}{2}$.

Exercice 16. On pose, pour tout $n \in \mathbb{N}^*$, $u_n = \displaystyle\sum_{k=1}^{n} \frac{1}{k^2}$ et $v_n = u_n + \frac{1}{n}$. Montrer que $(u_n)_n$ et $(v_n)_n$ sont adjacentes et convergent vers une limite commune l. *On admet que cette limite commune est $\dfrac{\pi^2}{6}$).*

En déduire un code Python permettant de fournir une approximation à 10^{-6} près de l.

Exercice 17. On pose, pour tout $n \in \mathbb{N}^*$, $u_n = (\displaystyle\sum_{k=1}^{n} \frac{1}{k}) - \ln(n)$ et $v_n = (\displaystyle\sum_{k=1}^{n} \frac{1}{k}) - \ln(n+1)$. Montrer que $(u_n)_n$ et $(v_n)_n$ sont adjacentes et convergent vers une limite commune γ.

En déduire un code Python permettant de fournir une approximation à 10^{-6} près de γ.