TP2 : Calcul du premier entier tel qu'une condition est vérifiée

(Révisions sur la structure itérative `while`*)*

▶ Dans votre dossier `Info_2a`, créez le dossier `TP_2`.

I. Introduction du problème

Le problème qui nous intéresse ici est le suivant.

Problème.

Données :

- Une suite (u_n) et une suite (d_n) telle que $d_n \underset{n\to+\infty}{\longrightarrow} 0$.
- L'existence d'un réel α tel que : $\forall n \in \mathbb{N},\ |u_n - \alpha| \leqslant d_n$.

But :

1) Déterminer un indice N tel que le terme u_N vérifie : $|u_N - \alpha| \leqslant 10^{-4}$.

2) En déduire une valeur approchée de α à 10^{-4} près.

Remarque

- Les inégalités du type $|u_n - \alpha| \leqslant d_n$ sont fréquentes dans les exercices. On pense notamment à l'utilisation de l'inégalité des accroissements finis pour l'étude des suites du type $u_{n+1} = f(u_n)$.
- La valeur de α n'est pas forcément connue précisément. C'est par exemple le cas lorsque α est fourni par le théorème de la bijection : on sait alors dans quel intervalle se situe α mais on ne connaît pas sa valeur exacte.
- Comme $|u_n - \alpha| \leqslant d_n$ et $d_n \underset{n\to+\infty}{\longrightarrow} 0$, on en conclut, à l'aide du théorème d'encadrement, que $|u_n - \alpha| \underset{n\to+\infty}{\longrightarrow} 0$ et donc que (u_n) est convergente, de limite α. Ceci démontre que l'élément N du point ***1)*** existe bien et que l'inégalité $|u_N - \alpha| \leqslant 10^{-4}$ est vérifiée à partir d'un certain rang.
- L'idée de base pour déterminer N est de calculer successivement les termes de (u_n) jusqu'à celui qui vérifie $|u_n - \alpha| \leqslant 10^{-4}$. Cependant, on ne peut procéder de la sorte si la valeur de α n'est pas connue (calcul de $u_n - \alpha$ impossible). On se sert alors de la suite (d_n).
  Si on parvient à déterminer un entier N tel que $d_N \leqslant 10^{-4}$, alors on obtient, par transitivité :

  $$|u_N - \alpha| \ \leqslant\ d_n \ \leqslant\ 10^{-4}$$

  ce qui permet de résoudre le problème.
- Une fois l'entier N précédent déterminé, u_N est une valeur approchée de α.

II. Un exemple classique

On commence par illustrer le problème et sa résolution par l'étude d'une suite de type $u_{n+1} = f(u_n)$ dans le cadre de l'utilisation de l'inégalité des accroissements finis.

On considère la fonction $f : x \mapsto \mathrm{e}^{-\frac{x^2}{2}}$ et on définit la suite (u_n) par : $\left\{ \begin{array}{l} u_0 = \frac{1}{2} \\ \forall n \in \mathbb{N},\ u_{n+1} = f(u_n) \end{array} \right.$

Rappelons les différentes étapes de ce type d'étude. Les démonstrations sont laissées au lecteur.

1) En appliquant le théorème de la bijection à la fonction $g : x \mapsto f(x) - x$, on démontre que l'équation $f(x) = x$ admet une unique solution dans $[0, 1]$, que l'on note α.

2) Après avoir démontré que l'intervalle $[0, 1]$ est stable par f, on en déduit, par récurrence que : $\forall n \in \mathbb{N},\ u_n \in [0, 1]$.

3) a) Par étude de la fonction f', on démontre : $\forall x \in [0, 1],\ |f'(x)| \leqslant \dfrac{1}{\sqrt{\mathrm{e}}}$.

b) On est alors dans le cadre de l'application de l'IAF, qui permet de démontrer que :

$$\forall n \in \mathbb{N},\ |u_{n+1} - \alpha| \leqslant \frac{1}{\sqrt{\mathrm{e}}} |u_n - \alpha|$$

(on démontre en fait que $|f(u_n) - f(\alpha)| \leqslant \dfrac{1}{\sqrt{e}} |u_n - \alpha|$*)*

c) On en déduit que : $\forall n \in \mathbb{N},\ |u_n - \alpha| \leqslant \left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n$.

d) Comme $\left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n \underset{n \to +\infty}{\longrightarrow} 0$, on en déduit que (u_n) est convergente, de limite α.

Le but est alors de calculer une valeur approchée de α à 10^{-4} près.

▶ Quelle condition permet d'assurer que $|u_n - \alpha| \leqslant 10^{-4}$?

> Si $\left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$, on en déduit par transitivité que $|u_n - \alpha| \leqslant \left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$.

▶ Écrire un programme permettant d'afficher le premier entier n tel que $\left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$.

```
import numpy as np
n = 0
while (1 / (np.sqrt(np.e)))**n > 10**(-4) :
    n = n + 1
print("La valeur de n est : " + str(n))
```

on peut améliorer cette version en calculant au fur et à mesure $\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n$:

```
n = 0
aux = 1
while aux > 10**(-4) :
    aux = aux * 1/(np.sqrt(np.e))
    n = n + 1
print("La valeur de n est : " + str(n))
```

- Compléter le programme précédent afin qu'il affiche une valeur approchée à 10^{-4} près de α.

```
n = 0
u = 1/2
while (1 / (np.sqrt(np.e)))**n > 10**(-4) :
    n = n + 1
    u = np.exp(-u**2 / 2)
print("La valeur de n est : " + str(n))
print("alpha peut être approché par : " + str(u))
```

(on aurait pu, comme précédemment, calculer $(\frac{1}{\sqrt{e}})^n$ par multiplications successives)

- Déterminer une formule mathématique donnant le premier entier n tel que $\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$.

$$\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4} \Leftrightarrow n\ln\left(\frac{1}{\sqrt{\mathrm{e}}}\right) \leqslant -4\ln(10) \quad \textit{(par stricte croissance de la fonction } \ln \textit{ sur } \mathbb{R}_+^*\textit{)}$$

$$\Leftrightarrow -n\ln\left(\sqrt{\mathrm{e}}\right) \leqslant -4\ln(10)$$

$$\Leftrightarrow n \geqslant \frac{\cancel{-}4\ln(10)}{\cancel{-}\ln\left(\sqrt{\mathrm{e}}\right)} = \frac{4\ln(10)}{\ln(\mathrm{e}^{\frac{1}{2}})} = \frac{4\ln(10)}{\frac{1}{2}\ln(\mathrm{e})} = 8\ln(10)$$

Ainsi, le premier entier tel que $\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$ est le premier entier tel que : $n \geqslant 8\ln(10)$.

L'entier cherché est donc $\lceil 8\ln(10) \rceil$.

- Comparer la valeur obtenue dans la quesiton précédente et celle affichée par le programme.

L'instruction `ceil(8 * log(10))` fournit bien le résultat 19 qui correspond à la valeur affichée par le programme.

- De même, donner la formule permettant d'obtenir le premier entier n tel que $\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant \varepsilon$.

Par une étude similaire, on trouve : $\lceil -2\ln(\varepsilon) \rceil$.

- En déduire une fonction `calcApproch` qui prend en paramètre un réel `eps` et qui calcule une valeur approchée de α à `eps` près à l'aide d'une boucle `for`. Le résultat sera stocké dans une variable `u`.

```
def calcApproch(eps) :
    u = 1/2
    n = np.ceil(-2 * np.log(eps))
    for i in range(n) :
        u = np.exp(-u**2 / 2)
    return u
```

III. Les exemples aux concours

Il est fréquent de devoir coder des programmes permettant de calculer un entier n / le premier entier n tel qu'une condition est vérifiée. On retrouve ce type d'exercice sous de nombreuses variantes.

III.1. EML 2018

L'épreuve `EML 2018` comportait une étude de suite récurrente d'ordre 1 (je ne peux y croire) (u_n) définie par :

$$\begin{cases} u_0 = 4 \\ \forall n \in \mathbb{N},\ u_{n+1} = \ln(u_n) + 2 \end{cases}$$

On devait démontrer les propriétés suivantes.

- $\forall n \in \mathbb{N},\ u_n \geqslant b$.
- $\forall n \in \mathbb{N},\ u_{n+1} - b \leqslant \dfrac{1}{2}\ (u_n - b)$.
- $\forall n \in \mathbb{N},\ 0 \leqslant u_n - b \leqslant \dfrac{1}{2^{n-1}}$.

▸ Écrire une fonction **Python** d'en-tête `def suite(n)` qui, prenant en argument un entier n de $\mathbb{N}$, renvoie la valeur de u_n.

On rappelle la fonction écrite lors du TP précédent.

```
1  import numpy as np
2  def suite(n) :
3      u = 4
4      for k in range(n) :
5          u = np.log(u) + 2
6      return u
```

▸ Recopier et compléter la ligne 3 de la fonction **Python** suivante afin que, prenant en argument un réel `epsilon` strictement positif, elle renvoie une valeur approchée de b à `epsilon` près.

```
1  def valeur_approchee(epsilon) :
2      n = 0
3      while ..... :
4          n = n + 1
5      b = suite(n)
6      return b
```

- On cherche ici à trouver un entier N tel que u_N est une valeur approchée de b à une précision ε près (fournie par l'utilisateur). Autrement dit, on souhaite exhiber $N \in \mathbb{N}$ tel que :
$$|u_N - b| \leqslant 10^{-3}$$
- Or, d'après la question ***6.b)*** : $\forall n \in \mathbb{N},\ 0 \leqslant u_n - b \leqslant \dfrac{1}{2^{n-1}}$.
- Il suffit alors de trouver $N \in \mathbb{N}$ tel que : $\dfrac{1}{2^{N-1}} \leqslant \varepsilon$.
Si c'est le cas, on obtient alors, par transitivité :
$$0 \leqslant u_N - b \leqslant \frac{1}{2^{N-1}} \leqslant \varepsilon$$
- On complète alors le programme **Python** de la façon suivante :

```
3        while 1 / 2**(n-1) > epsilon :
```

À la fin de la boucle, on est assuré que : $\dfrac{1}{2^{n-1}} \leqslant \varepsilon$ (on itère tant que ce n'est pas le cas).

Il reste alors à calculer la valeur approchée de b : on l'obtient par le caclul de u_n où n est la valeur obtenue à l'issue de cette boucle.

```
6        b = suite(n)
```

Commentaire

- Lorsqu'on écrit une boucle `while` il est préférable de s'assurer en amont de sa terminaison. C'est bien le cas ici. En effet, la suite $(\frac{1}{2^{n-1}})_{n\geqslant 1}$ est convergente de limite 0. Ce qui signifie :
$$\forall \varepsilon > 0,\ \exists n_0 \in \mathbb{N},\ \forall n \geqslant n_0,\ \left|\frac{1}{2^{n-1}} - 0\right| < \varepsilon$$
Ainsi, quelle que soit la précision $\varepsilon > 0$ choisie au départ, on est toujours en mesure de trouver un rang n_0 à partir duquel on aura : $\dfrac{1}{2^{n-1}} < \varepsilon$.
- On pouvait déterminer, sans utiliser de boucle, un entier N tel que u_N est une valeur approchée à ε près de b. Pour ce faire, on remarque :
$$\frac{1}{2^{n-1}} \leqslant \varepsilon \quad\Leftrightarrow\quad 2^{n-1} \geqslant \frac{1}{\varepsilon} \quad\Leftrightarrow\quad (n-1)\ \ln(2) \geqslant \ln\left(\frac{1}{\varepsilon}\right) \quad\Leftrightarrow\quad (n-1)\ \geqslant \frac{-\ln(\varepsilon)}{\ln(2)}$$
L'entier $N = \left\lceil \dfrac{-\ln(\varepsilon)}{\ln(2)} \right\rceil$ convient.

III.2. EML 2019

L'épreuve `EML 2019` comportait une étude de suite récurrente (ça a dû déstabiliser les candidats !) (u_n) définie par :

$$\begin{cases} u_1 = 1 \\ \forall n \in \mathbb{N}^*,\ u_{n+1} = u_n + \dfrac{1}{n^2\, u_n} = \dfrac{1}{n}\, f(n\, u_n) \end{cases}$$

où $f : t \mapsto t + \dfrac{1}{t}$.

On démontrait les propriétés suivantes.

- (u_n) converge vers $\ell \in [2, 3]$.
- $\forall p \geqslant 2,\ 0 \leqslant \ell - u_p \leqslant \dfrac{1}{p-1}$.

▸ En déduire une fonction **Python** qui renvoie une valeur approchée de ℓ à 10^{-4} près.

- On cherche ici à trouver un entier N tel que u_N est une valeur approchée de ℓ à 10^{-4} près. Autrement dit, on souhaite exhiber $N \in \mathbb{N}$ tel que :

$$|\ell - u_N| \ \leqslant\ 10^{-4}$$

- Or, d'après la question précédente : $\forall p \geqslant 2, \quad 0 \ \leqslant\ \ell - u_p \ \leqslant\ \dfrac{1}{p-1}$.
- Il suffit alors de trouver $N \in \mathbb{N}$ tel que : $\dfrac{1}{N-1} \leqslant 10^{-4}$.

  Si c'est le cas, on obtient alors par transitivité :

$$0 \ \leqslant\ \ell - u_N \ \leqslant\ 10^{-4}$$

- On propose alors le programme suivant :

```
1  def valeur_approchee() :
2      n = 2
3      while 1 / (n-1) > 10**(-4) :
4          n = n + 1
5      l = suite(n)
6      return l
```

Détaillons les éléments de ce script.

× **Début du script**

La variable `n` est initialisée à 2. En effet, on souhaite pouvoir effectuer le calcul : $\dfrac{1}{\texttt{n}-1}$.

```
2      n = 2
```

× **Structure itérative**

Les lignes 3 à 4 consistent à déterminer le plus petit entier n tel que $\dfrac{1}{n-1} \leqslant 10^{-4}$. On doit donc comparer les valeurs successives de la suite $\left(\dfrac{1}{n-1}\right)_{n\geqslant 2}$ au réel 10^{-4} jusqu'à ce que $\dfrac{1}{n-1} \leqslant 10^{-4}$. Autrement dit, on doit comparer ces valeurs successives à 10^{-4} tant que $\dfrac{1}{n-1} > 10^{-4}$. Pour cela on met en place une structure itérative (boucle while) :

```
3        while 1 / (n-1) > 10**(-4) :
```

On met alors à jour en conséquence la variable `n` : on ajoute 1 pour signaler qu'on va comparer le terme suivant de la suite $\left(\dfrac{1}{n-1}\right)_{n\geqslant 2}$ à 10^{-4}.

```
4            n = n + 1
```

× **Fin du script**

À la fin de cette boucle, on est assuré que : $\dfrac{1}{n-1} \leqslant 10^{-4}$ (on itère tant que ce n'est pas le cas).
Il reste alors à calculer la valeur approchée de ℓ : on l'obtient par le calcul de u_n où n est la valeur obtenue à l'issue de cette boucle.

```
6        l = suite(n)
```

Commentaire

- Lorsqu'on écrit une boucle `while`, il est préférable de s'assurer en amont de sa terminaison. C'est bien le cas ici. En effet, la suite $\left(\dfrac{1}{n-1}\right)_{n\geqslant 2}$ est convergente de limite 0. Ce qui signifie :

$$\forall \varepsilon > 0,\ \exists n_0 \in \mathbb{N},\ \forall n \geqslant n_0,\ \left|\frac{1}{n-1} - 0\right| < \varepsilon$$

  Ainsi, quelle que soit la précision $\varepsilon > 0$ choisie au départ (ici 10^{-4}), on est toujours en mesure de trouver un rang n_0 à partir duquel on aura : $\dfrac{1}{n-1} < 10^{-4}$.
- On pouvait déterminer, sans utiliser de boucle, un entier N tel que u_N est une valeur approchée à 10^{-4} près de ℓ. Pour ce faire, on remarque :

$$\frac{1}{n-1} \leqslant 10^{-4} \ \Leftrightarrow\ n-1 \geqslant 10^4 \ \Leftrightarrow\ n \geqslant 10^4 + 1$$

  L'entier $N = \lceil 10^4 + 1 \rceil$ convient.

III.3. ECRICOME 2018

Pour tout entier naturel n non nul, on pose : $u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.

On démontrait dans cet exercice que la suite (u_n) était convergente, vers une limite notée $\gamma \in \mathbb{R}$ puis :

$$\forall n \in \mathbb{N}^*,\ |u_n - \gamma| \leqslant \frac{1}{n}$$

▸ On rappelle que l'instruction `np.floor(x)` renvoie la partie entière d'un réel x et on suppose que la fonction `u` de la question ***1.e)*** a été correctement programmée. Expliquer l'intérêt et le fonctionnement du script ci-dessous :

```
1  eps = float(input("Entrer un réel strictement positif : "))
2  n = int(np.floor(1/eps) + 1)
3  print(u(n))
```

- Ce script a pour but d'afficher une valeur approchée de γ à ε près (où ε est un réel strictement positif fourni par l'utilisateur et stocké dans la variable `eps`).
  Pour ce faire, il faut commencer par trouver un entier $N \in \mathbb{N}^*$ tel que :
  $$\left|u_N - \gamma\right| \leqslant \varepsilon$$
- Or, d'après ce qui précède : $\forall n \in \mathbb{N}^*,\ \left|u_n - \gamma\right| \leqslant \frac{1}{n}$.
- Afin de trouver l'entier N recherché, il suffit de trouver un entier $N \in \mathbb{N}^*$ tel que :
  $$\frac{1}{N} \leqslant \varepsilon$$
  Si c'est le cas, on obtient alors, par transitivité :
  $$\left|u_N - \gamma\right| \leqslant \frac{1}{N} \leqslant \varepsilon$$
- Raisonnons par équivalence pour trouver N :
  $$\frac{1}{n} \leqslant \varepsilon \quad \Leftrightarrow \quad n \geqslant \frac{1}{\varepsilon} \qquad \textit{(par décroissance de la fonction inverse sur }]0,+\infty[\textit{)}$$
  Ainsi, tout entier plus grand que $\frac{1}{\varepsilon}$ convient. En particulier, l'entier $N = \lfloor \frac{1}{\varepsilon} \rfloor + 1$ convient.

Ce script affiche la valeur u_N où $N = \lfloor \frac{1}{\varepsilon} \rfloor + 1$. C'est une valeur approchée de γ à ε près.

III.4. ECRICOME 2019

Pour tout entier n non nul, on note h_n la fonction définie sur $\mathbb{R}_+^*$ par :

$$\forall x > 0,\quad h_n(x) \;=\; f(x^n, 1) \;=\; x^n + 1 + \frac{1}{x^n}$$

On démontrait les propriétés suivantes.

- Pour tout entier naturel n non nul, la fonction h_n est strictement décroissante sur $]0,1[$ et strictement croissante sur $[1,+\infty[$.
- Pour tout entier n non nul, l'équation $h_n(x) = 4$ admet exactement deux solutions, notées u_n et v_n et vérifiant : $0 < u_n < 1 < v_n$.

▸ Écrire une fonction **Python** d'en-tête `def h(n,x)` qui renvoie la valeur de $h_n(x)$ lorsqu'on lui fournit un entier naturel n non nul et un réel $x \in \mathbb{R}_+^*$ en entrée.

```
def h(n, x) :
    y = x**n + 1 + (1 / x**n )
    return y
```

▸ Compléter la fonction suivante pour qu'elle renvoie une valeur approchée à 10^{-5} près de v_n par la méthode de dichotomie lorsqu'on lui fournit un entier $n \geqslant 1$ en entrée :

```
def v(n) :
    a = 1
    b = 3
    while (b-a) > 10**(-5) :
        c = (a+b) / 2
        if h(n,c) < 4 :
            ........
        else :
            ........
    return ..........
```

Commençons par rappeler le cadre de la recherche par dichotomie.

Calcul approché d'un zéro d'une fonction par dichotomie

Données :

× une fonction $f : \mathbb{R} \to \mathbb{R}$,

× un intervalle de recherche $[a, b]$,

× une précision de recherche ε.

Résultat : une valeur approchée à ε près d'un zéro (sur l'intervalle $[a, b]$) de la fonction f.

Autrement dit, une valeur approchée (à ε près) d'un réel $x \in [a, b]$ tel que : $f(x) = 0$.

- La dichotomie est une méthode itérative dont le principe, comme son nom l'indique, est de découper à chaque itération l'intervalle de recherche en deux nouveaux intervalles. L'intervalle de recherche est découpé en son milieu. On obtient deux nouveaux intervalles :

  × un intervalle dans lequel on sait que l'on va trouver un zéro de f.
  Cet intervalle est conservé pour l'itération suivante.

  × un intervalle dans lequel ne se trouve pas forcément un zéro de f.
  Cet intervalle n'est pas conservé dans la suite de l'algorithme.

  La largeur de l'intervalle de recherche est ainsi divisée par 2 à chaque itération.
  On itère jusqu'à obtenir un intervalle I contenant un zéro de f et de largeur plus faible que ε.
  Les points de cet intervalle I sont tous de bonnes approximations du zéro contenu dans I.

- C'est le **théorème des valeurs intermédiaires** qui permet de choisir l'intervalle qu'il faut garder à chaque étape. Rappelons son énoncé et précisons maintenant l'algorithme :

Théorème des Valeurs Intermédiaires

Soit $f : [a, b] \to \mathbb{R}$ continue sur l'**intervalle** $[a, b]$.

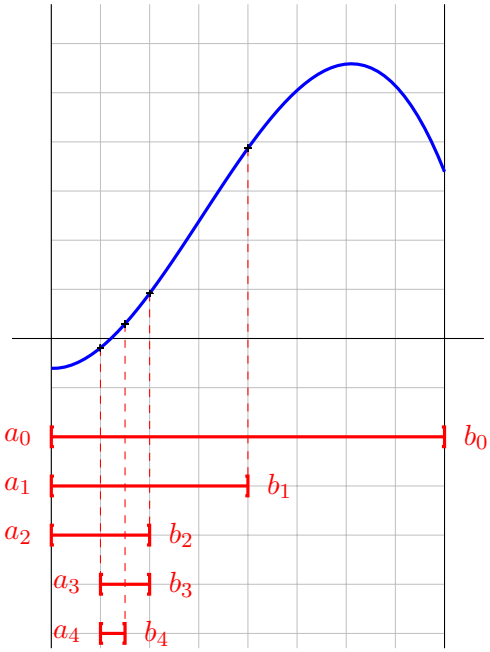
Supposons : $f(a)\ f(b) \leqslant 0$.

Alors il existe $c \in [a, b]$ tel que $f(c) = 0$.

Calcul des suites (a_m), (b_m), (c_m)

Cas $f(a) \leqslant 0$ et $f(b) \geqslant 0$

- Initialement, $a_0 = a$, $b_0 = b$
- À chaque tour de boucle (tant que $b_m - a_m > \varepsilon$) :
  - $c_m = \dfrac{a_m + b_m}{2}$ (point milieu de $[a_m, b_m]$)
    - si $f(c_m) < 0$ alors :
      - $a_{m+1} = c_m$
      - $b_{m+1} = b_m$
    - si $f(c_m) \geqslant 0$ alors :
      - $a_{m+1} = a_m$
      - $b_{m+1} = c_m$



- On construit ainsi une suite $\big([a_m, b_m]\big)_{m\in\mathbb{N}}$ de segments emboîtés :
  - contenant tous un zéro de f,
  - dont la largeur est divisée par deux d'un rang au suivant.

Soit $n \in \mathbb{N}^*$. On cherche une valeur de x telle que : $h_n(x) = 4$ ce qui s'écrit :

$$h_n(x) - 4 = 0 \quad \text{ou encore} \quad f_n(x) = 0 \quad \text{où} \quad f_n : x \mapsto h_n(x) - 4$$

On se fixe initialement l'intervalle de recherche $[1, 3]$ de sorte que l'équation $f_n(x) = 0$ ne possède qu'une solution, à savoir la valeur v_n qu'on cherche à approcher. D'un point de vue informatique, on crée des variables `a` et `b` destinées à contenir les valeurs succesives de a_m et b_m. Ces variables sont initialisées respectivement à 1 et 3.

```
2    a = 1
3    b = 3
```

On procède alors de manière itérative, tant que l'intervalle de recherche n'est pas de largeur plus faible que la précision 10^{-5} escomptée.

```
4    while (b-a) > 10**(-5) :
```

On commence par définir le point milieu du segment de recherche.

```
5        c = (a+b) / 2
```

Puis on teste si $f_n(\texttt{c}) < 0$, c'est-à-dire si $h_n(\texttt{c}) < 4$.
Si c'est le cas, la recherche s'effectue dans le demi-segment de droite.

```
6        if h(n,c) < 4 :
7            a = c
```

Sinon, elle s'effectue dans le demi-segment de gauche.

```
8           else :
9               b = c
```

En sortie de boucle, on est assuré que le segment de recherche, mis à jour au fur et à mesure de l'algorithme, est de largeur plus faible que 10^{-5} et contient un zéro de f_n. Tout point de cet intervalle est donc une valeur approchée à 10^{-5} près de ce zéro.
On peut alors choisir de renvoyer le point le plus à gauche du segment.

```
10      return a
```

On peut tout aussi bien choisir le point le plus à droite :

```
10      return b
```

Ou encore le point milieu :

```
10      return (a + b) / 2
```

Ce dernier choix présente un avantage : tout point (dont le zéro recherché) du dernier intervalle de recherche se situe à une distance d'au plus $\dfrac{10^{-5}}{2}$ de ce point milieu.
On obtient ainsi une valeur approchée à $\dfrac{10^{-5}}{2}$ du zéro recherché.

Commentaire

- On peut se demander combien de tours de boucle sont nécessaires pour obtenir le résultat. Pour le déterminer, il suffit d'avoir en tête les éléments suivants :
  - × l'intervalle de recherche initial $[1, 3]$ est de largeur 2.
  - × la largeur de l'intervalle de recherche est divisée par 2 à chaque tour de boucle. À la fin du $m^{\text{ème}}$ tour de boucle, l'intervalle de recherche est donc de largeur $\dfrac{2}{2^m}$.
  - × l'algorithme s'arrête lorsque l'intervalle devient de largeur plus faible que 10^{-5}.

  On obtient le nombre d'itérations nécessaires en procédant par équivalence :

  $$\frac{2}{2^m} \leqslant 10^{-5} \;\Leftrightarrow\; \frac{2^m}{2} \geqslant 10^5 \;\Leftrightarrow\; 2^m \geqslant 2 \times 10^5 \;\Leftrightarrow\; m\ \ln(2) \geqslant \ln(2) + 5\ \ln(10)$$

  Ainsi : $\left\lceil 5\ \dfrac{\ln(10)}{\ln(2)} + 1 \right\rceil$ tours de boucle suffisent.

  On retiendra que si l'on souhaite obtenir une précision de 5 chiffres après la virgule, il suffit d'effectuer de l'ordre de 5 tours de boucle. Cette algorithme est donc extrêmement rapide.
- Afin de permettre une bonne compréhension des mécanismes en jeu, on a détaillé avec beaucoup de précision la réponse à cette question. Cependant, compléter correctement le programme **Python** (on place ci-dessous le programme obtenu) démontre la bonne compréhension de l'algorithme demandé et permet d'obtenir tous les points alloués à cette question.

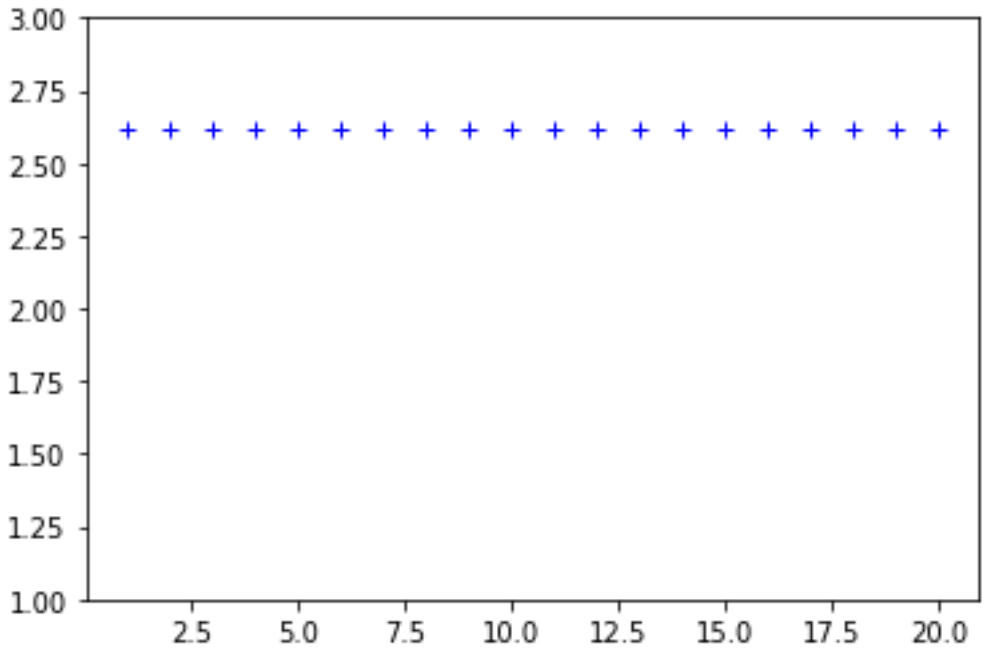
- On obtient le programme complet suivant.

```
def v(n) :
    a = 1
    b = 3
    while (b-a) > 10**(-5) :
        c = (a+b) / 2
        if h(n,c) < 4 :
            a = c
        else :
            b = c
    return (a + b) / 2
```

▸ À la suite de la fonction v, on écrit le code suivant :

```
import matplotlib.pyplot as plt
X = [k for k in range(1,21)]
Y = []
for k in range(1,21) :
    Y.append( v(k)**k )
plt.plot(X, Y, ‘‘b+‘‘)
ply.ylim(1, 3)
```

À l'exécution du programme, on obtient la sortie graphique suivante :



Expliquer ce qui est affiché sur le graphique ci-dessus.
Que peut-on conjecturer ?

- Le programme commence par définir deux listes `X` et `Y`.

```
X = [k for k in range(1,21)]
Y = []
```

La liste `X` contient initialement $[1, 2, \ldots, 20]$, c'est-à-dire les 20 premiers entiers non nuls.
La liste `Y` est destinée à contenir les 20 premières valeurs de la suite $(v_n{}^n)$.
Il est initialement vide.

- À l'aide d'une boucle, le $k^{\text{ème}}$ élément de la liste `Y` contient une valeur approchée de v_k^k.

```
for k in range(1,21) :
    Y.append( v(k)**k )
```

- On effectue alors le tracé des points d'abscisse une valeur de `X` et d'ordonnée la valeur correspondante de `Y`. On obtient ainsi le tracé des points de coordonnées (k, v_k^k) pour k variant de 1 à 20.

```
plt.plot(X, Y, "b+")
ply.ylim(1, 3)
```

Le nuage de points obtenu correspond au tracé des 20 premières valeurs de la suite $(v_n{}^n)$. Au vu de la représentation graphique obtenue, on peut faire la conjecture que la suite $(v_n{}^n)$ est constante et approximativement de valeur 2.61.

III.5. ESSEC-I 2020

Cet énoncé introduisait, pour tout $n \in \mathbb{N}^*$ un événement G_n et faisait démontrer les résultats suivants :

$$\mathbb{P}(G_1) = \frac{p}{q} - \left(1 - \frac{p}{q}\right) pq \qquad \text{et} \qquad \mathbb{P}(G_{n+1}) = \mathbb{P}(G_n) - \left(1 - \frac{p}{q}\right) (pq)^{n+1} \binom{2n+1}{n+1}$$

où $q = 1 - p$.
Après avoir codé une fonction `CoeffBin` permettant de calculer les coefficients binomiaux (voir TP1), l'énoncé demandait d'écrire un script **Python** qui déterminer n_p, le plus petit entier n tel que $\mathbb{P}(G_n) \leqslant \varepsilon$ pour $p < \dfrac{1}{2}$ et $\varepsilon > 0$ saisis au clavier par l'utilisateur.

On propose alors le script Scilab suivant.

```
p = float(input("Entrez la valeur de p : ")
eps = float(input("Entrez la valeur de epsilon : ")
q = 1 - p
n = 1
ProbGn = p/q - (1 - p/q) * p * q
while ProbGn > eps :
    ProbGn = ProbGn - (1 - p/q) * (p*q)**(n+1) * CoeffBin(n+1, 2*n + 1)
    n = n + 1
print(n)
```

Détaillons les éléments de ce programme.

- **Début du programme**
  On commence par demander à l'utilisateur d'entrer une valeur pour le paramètre `p` et pour la précision `eps`.

```
1  p = float(input(‘‘Entrez la valeur de p : ‘‘)
2  eps = float(input(‘‘Entrez la valeur de epsilon : ‘‘)
```

  On définit la variable `q`.

```
3  q = 1 - p
```

  La variable `n` est initialisée à 1.
  La variable `ProbGn`, qui contiendra les valeurs successives de la suite $\big(\mathbb{P}(G_n)\big)_{n\in\mathbb{N}^*}$, est initialisée à $\mathbb{P}(G_1)$ (calculée en question ***15.d)***).

```
4  n = 1
5  ProbGn = p/q - (1 - p/q) * p * q
```

- **Structure itérative**
  Les lignes 6 à 8 consistent à déterminer le plus petit entier n tel que : $\mathbb{P}(G_n) \leqslant \varepsilon$. On doit donc calculer les valeurs successives de la suite $\big(\mathbb{P}(G_n)\big)$ jusqu'à ce que $\mathbb{P}(G_n) \leqslant \varepsilon$. Autrement dit, on doit calculer ces valeurs successives tant que $\mathbb{P}(G_n) > \varepsilon$. Pour cela, on met en place une structure itérative (boucle while).

```
6  while ProbGn > eps :
```

  Tant que $\mathbb{P}(G_n) > \varepsilon$, on calcule $\mathbb{P}(G_{n+1})$ et on stocke toujours cette valeur dans `ProbGn` (on utilise ici la formule de la question ***15.c)***) :

```
7      ProbGn = ProbGn - (1 - p/q) * (p*q)**(n+1) * CoeffBin(n+1, 2*n +1)
```

  On met alors à jour en conséquence la variable `n` : on ajoute 1 pour signaler qu'on a calculé $\mathbb{P}(G_{n+1})$.

```
8          n = n + 1
```

- **Fin du programme**
  À l'issue de cette boucle, la variable `n` contient le plus petit entier n tel que $\mathbb{P}(G_n) \leqslant \varepsilon$. On affiche alors enfin la valeur de la variable `n`.

```
9  print(n)
```