# Méthodes itératives : Annales 2015 / 2016 / 2017 / 2018 / 2019

## I. Suites de type $u_{n+1} = f(u_n)$

ECRICOME – 2015

- On considère la fonction $F$ définie par :

$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 - e^{-x} & \text{si } x \geqslant 0 \end{cases}$$

*(fonction de répartition de d'une v.a.r. $X$ telle que $X \hookrightarrow \mathcal{E}(1)$)*
et on définit la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par :

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

***a)*** Recopier et compléter le programme **Scilab** suivant qui permet de représenter les cent premiers termes de la suite $(u_n)_{n\in\mathbb{N}^*}$ :

```
1  U = zeros(1,100)
2  U(1) = 1
3  for n = 1 : 99
4      U(n+1) = ------
5  end
6  plot(U,"+")
```

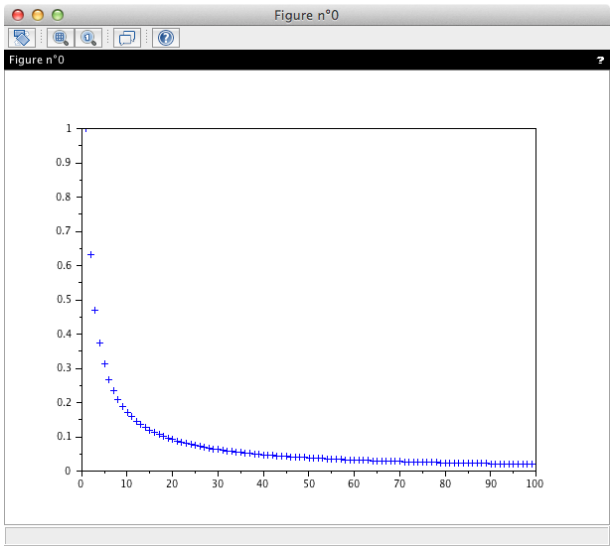
*Démonstration.*

```
1  U = zeros(1,100)
2  U(1) = 1
3  for n = 1 : 99
4      U(n+1) = 1 - exp( U(n) )
5  end
6  plot(U,"+")
```

□

***b)*** Le programme complété permet d'obtenir la représentation graphique suivante.
Quelle conjecture pouvez-vous émettre sur la monotonie et la limite de la suite $(u_n)_{n\in\mathbb{N}^*}$ ?

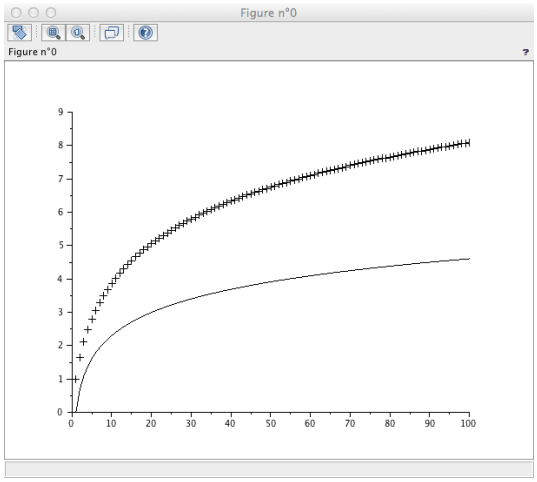
*Démonstration.*
D'après la représentation graphique, la suite $(u_n)$ semble être convergente, de limite nulle. □




***c)*** On modifie le programme écrit en question ***a)*** en remplaçant la dernière ligne par :

```
1  X = 1:100
2  S = cumsum(U)
3  Y = log(X)
4  plot2d(X, S, -1)
5  plot2d(X, Y)
```

Le programme ci-dessus permet d'obtenir la représentation graphique suivante :




Que représente le vecteur-ligne `S` ?
Quelle conjecture pouvez-vous émettre sur la nature de la série de terme général $u_n$ ?

*Démonstration.*

- Le vecteur `U` contient les 100 premiers éléments de la suite $(u_n)$. L'opérateur `cumsum` permet de calculer la somme cumulée de ce vecteur. Ainsi, `S` contient les 100 premières sommes partielles de la série $\sum\ u_n$.
- D'après le tracé obtenu, on peut émettre l'hypothèse que la série $\sum\ u_n$ est divergente. Plus précisément, que $S_n \underset{n\to+\infty}{\longrightarrow} +\infty$.

□

ECRICOME – 2018

- On considère les matrices : $A = \begin{pmatrix} 2 & 1 & -2 \\ 0 & 3 & 0 \\ 1 & -1 & 5 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & -1 & -1 \\ -3 & 3 & -3 \\ -1 & 1 & 1 \end{pmatrix}$.

  On pose $X_0 = \begin{pmatrix} 3 \\ 0 \\ -1 \end{pmatrix}$, $X_1 = \begin{pmatrix} 3 \\ 0 \\ -2 \end{pmatrix}$, et pour tout $n \in \mathbb{N}$ : $X_{n+2} = \frac{1}{6} A X_{n+1} + \frac{1}{6} B X_n$.

***a)*** Compléter la fonction ci-dessous qui prend en argument un entier $n$ supérieur ou égal à 2 et qui renvoie la matrice $X_n$ :

```
1  function res = X(n)
2      Xold = [3; 0; -1]
3      Xnew = [3; 0; -2]
4      A = [2,1,-2; 0,3,0; 1,-1,5]
5      B = [1,-1,-1; -3,3,-3; -1,1,1]
6      for i = 2:n
7          Aux = ..........
8          Xold = .........
9          Xnew = .........
10     end
11     res = .........
12 endfunction
```

*Démonstration.*
Détaillons les différents éléments présents dans cette fonction.

- En ligne 2 et 3, on définit les variables `Xold` et `Xnew`.
  Ces deux variables sont initialement affectées aux valeurs $X_0$ et $X_1$.
- En ligne 4 et 5, on stocke les matrices $A$ et $B$ dans les variables `A` et `B`.
- De la ligne 6 à la ligne 10, on met à jour les variables `Xold` et `Xnew` de sorte à ce qu'elles contiennent les valeurs successives de la suite matricielle $(X_n)$.

```
6      for i = 2:n
7          Aux = Xold
8          Xold = Xnew
9          Xnew = 1/6 * A * Xold + 1/6 * B * Aux
10     end
```

Pour ce faire, on a introduit une variable auxiliaire `Aux`.
Détaillons le principe de cette boucle :

× avant le 1er tour de boucle :

`Xold` contient $X_0$ et `Xnew` contient $X_1$

lors du 1er tour de boucle (`i` contient 2) :

`Aux = Xold` — (`Aux` contient $X_0$, dernière valeur de `Xold`)

`Xold = Xnew` — (`Xold` contient $X_1$, dernière valeur de `Xnew`)

`Xnew = 1/6 * A * Xold + 1/6 * B * Aux` — (`Xnew` contient $X_2$, valeur obtenue par la définition $X_2 = \frac{1}{6} AX_1 + \frac{1}{6} BX_0$)

× avant le $2^{\text{ème}}$ tour de boucle, d'après ce qui précède :

`Xold` contient $X_0$, `Xold` contient $X_1$ et `Xnew` contient $X_2$

lors du $2^{\text{ème}}$ tour de boucle (`i` contient 3) :

| | |
|---|---|
| `Aux = Xold` | (`Aux` contient $X_1$, dernière valeur de `Xold`) |
| `Xold = Xnew` | (`Xold` contient $X_2$, dernière valeur de `Xnew`) |
| `Xnew = 1/6 * A * Xold + 1/6 * B * Aux` | (`Xnew` contient $X_3$, valeur obtenue par la définition $X_3 = \frac{1}{6}\,AX_2 + \frac{1}{6}\,BX_1$) |

× ...

× avant le $(n-1)^{\text{ème}}$ tour de boucle :

`Xold` contient $X_{n-3}$, `Xold` contient $X_{n-2}$ et `Xnew` contient $X_{n-1}$

lors du $(n-1)^{\text{ème}}$ tour de boucle (`i` contient $n$) :

| | |
|---|---|
| `Aux = Xold` | (`Aux` contient $X_{n-2}$, dernière valeur de `Xold`) |
| `Xold = Xnew` | (`Xold` contient $X_{n-1}$, dernière valeur de `Xnew`) |
| `Xnew = 1/6 * A * Xold + 1/6 * B * Aux` | (`Xnew` contient $X_n$, valeur obtenue par la définition $X_n = \frac{1}{6}\,AX_{n-1} + \frac{1}{6}\,BX_{n-2}$) |

– Enfin, il n'y a plus qu'à affecter à `res` la valeur $X_n$.

```
11        res = Xnew
```

**Commentaire**

- Afin de permettre une bonne compréhension des mécanismes en jeu, on a détaillé la réponse à cette question. Cependant, écrire correctement la fonction **Scilab** démontre la bonne compréhension et permet certainement d'obtenir tous les points alloués.
- On a démontré dans cette question que si, avant le $i^{\text{ème}}$ tour de boucle :

  `Aux` contient $X_{i-2}$, `Xold` contient $X_{i-1}$ et `Xnew` contient $X_i$

  alors, à l'issue de ce tour de boucle :

  `Aux` contient $X_{i-1}$, `Xold` contient $X_i$ et `Xnew` contient $X_{i+1}$

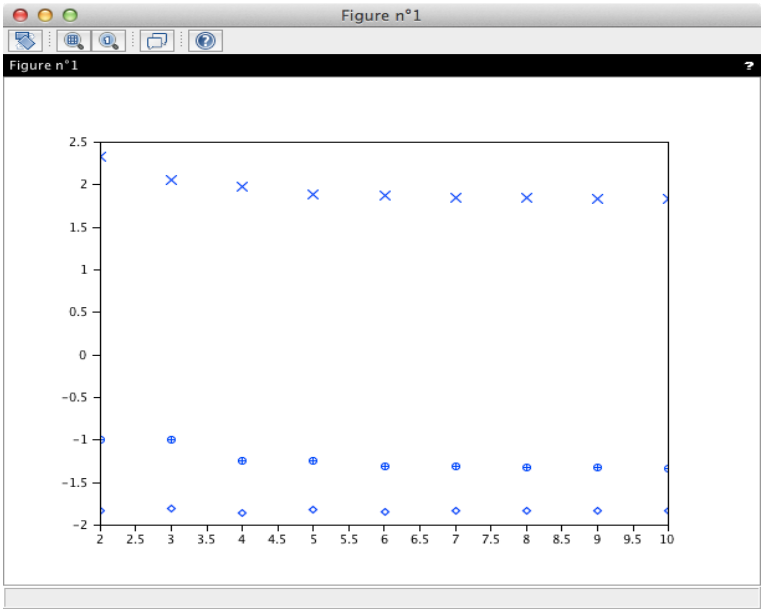
  Cette propriété est ce qu'on appelle un **invariant de boucle**. Elle permet d'assurer la correction de la fonction implémentée et notamment le fait qu'à l'issue du dernier tour de boucle la variable `Xnew` contient $X_n$.
- Il est à noter que si on teste la fonction avec une valeur de `n` strictement inférieure à 2, alors on n'entre pas dans la boucle (l'instruction `2:n` crée une matrice ligne vide). Dans ce cas, la variable `Xnew` n'est pas mise à jour et la variable `res` contient à la fin du programme $X_1$, soit la valeur initialement affectée à `Xnew`. Ainsi, la fonction renvoie la bonne valeur aussi lorsque la variable `n` prend la valeur 1.

- Dans l'exercice, il était noté $X_n = \begin{pmatrix} \alpha_n \\ \beta_n \\ \gamma_n \end{pmatrix}$ et on démontrait, pour tout $n \in \mathbb{N}$ :

$$\alpha_n = \frac{11}{6} + \frac{2}{3}\left(-\frac{1}{2}\right)^n + \left(\frac{1}{2}\right)^{n-1} - \frac{3}{2}\left(-\frac{1}{3}\right)^n$$

$$\beta_n = \left(\frac{1}{2}\right)^{n-1} - \frac{2}{3}\left(-\frac{1}{2}\right)^n - \frac{4}{3} \quad \text{et} \quad \gamma_n = -\frac{11}{6} - \frac{2}{3}\left(-\frac{1}{2}\right)^n + \frac{3}{2}\left(-\frac{1}{3}\right)^n$$

***b)*** La fonction précédente a été utilisée dans un script permettant d'obtenir graphiquement (voir figure 1) les valeurs de $\alpha_n$, $\beta_n$ et $\gamma_n$ en fonction de $n$.
Associer chacune des trois représentations graphiques à chacune des suites $(\alpha_n)_{n\in\mathbb{N}}$, $(\beta_n)_{n\in\mathbb{N}}$, $(\gamma_n)_{n\in\mathbb{N}}$ en justifiant votre réponse.




*Démonstration.*
Soit $n \in \mathbb{N}$.

- D'après la question ***5*** :

$$\beta_n = \left(\frac{1}{2}\right)^{n-1} - \frac{2}{3}\left(-\frac{1}{2}\right)^n - \frac{4}{3} \underset{n\to+\infty}{\longrightarrow} -\frac{4}{3} \simeq -1,33$$

En effet, comme $\left|\frac{1}{2}\right| < 1$ et $\left|-\frac{1}{2}\right| < 1$ alors : $\lim\limits_{n\to+\infty} \left(\frac{1}{2}\right)^{n-1} = 0$ et $\lim\limits_{n\to+\infty} \left(-\frac{1}{2}\right)^n = 0$.

- On démontre de même, toujours d'après les expressions obtenues en question ***5.*** :

$$\alpha_n \underset{n\to+\infty}{\longrightarrow} \frac{11}{6} \simeq 1,8 \quad \text{et} \quad \gamma_n \underset{n\to+\infty}{\longrightarrow} -\frac{11}{6} \simeq -1,8$$

- Or, d'après la figure :
  - × la suite repérée par × semble converger vers un réel valant approximativement $1,8$.

    Cette représentation graphique correspond à la suite $(\alpha_n)$.
  - × la suite repérée par $\oplus$ semble converger vers un réel valant approximativement $-1,3$.

Cette représentation graphique correspond à la suite $(\beta_n)$.

× la suite repérée par ◊ semble converger vers un réel valant approximativement $-1,8$.

Cette représentation graphique correspond à la suite $(\gamma_n)$.

**Commentaire**

Il est possible de rédiger autrement.
La représentation graphique commençant à 2, on peut calculer :

$$\alpha_2 = \frac{14}{6}, \quad \beta_2 = -1 \quad \text{et} \quad \gamma_2 = -\frac{11}{6}$$

ce qui permet d'associer chaque représentation graphique à la bonne suite.

□

EDHEC – 2019

- On définit la suite $(u_n)$ par : $\forall n \in \mathbb{N}$, $u_n = \dfrac{4^n\,(n!)^2}{(2n+1)!}$.

  On admet que, si `t` est un vecteur, la commande `prod(t)` renvoie le produit des éléments de `t`.
  Compléter le script **Scilab** suivant afin qu'il permette de calculer et d'afficher la valeur de $u_n$ pour une valeur de $n$ entrée par l'utilisateur.

```
1 n = input('entrez une valeur pour n : ')
2 x = 1:n
3 m = 2 * n + 1
4 y = 1:m
5 v = ..........
6 w = ..........
7 u = .......... * v^2 / w
8 disp(u)
```

*Démonstration.*

- Commentons tout d'abord le début du programme proposé.
  - × On commence par demander à l'utilisateur d'entrer une valeur pour l'entier `n`.

```
1 n = input('entrez une valeur pour n :')
```

  - × On stocke ensuite dans la variable `x` la matrice ligne contenant les entiers de 1 à `n`.

```
2 x = 1:n
```

  - × On stocke de plus dans la variable `m` la quantité $2\mathtt{n}+1$.

```
3 m = 1 * n + 1
```

  - × On stocke enfin dans la variable `y` la matrice ligne contenant les entiers de 1 à `m`, c'est-à-dire les entiers de 1 à $2\mathtt{n}+1$.

```
4 y = 1:m
```

- On cherche maintenant à stocker dans la variable `u` la valeur de $u_{\mathtt{n}}$.
  D'après la question ***5.b)***, pour tout $n \in \mathbb{N}$ :

$$u_n = \frac{4^n\,(n!)^2}{(2n+1)!}$$

× L'énoncé propose de compléter la commande suivante :

```
7  u = .......... * v^2 / w
```

Par analogie avec la formule de la question ***5.b)***, on souhaite donc stocker dans la variable `v` la valeur `n!` et dans la variable `w` la valeur $(2\mathtt{n}+1)!$.

× On cherche alors une commande permettant d'obtenir $\mathtt{n}! = 1 \times \cdots \times \mathtt{n}$.
D'après l'énoncé, la commande `prod(t)` renvoie le produit des éléments de `t`.
La variable `x` contient les entiers de 1 à `n`. Ainsi, pour obtenir `n!`, on peut utiliser la commande : `prod(x)`. On obtient donc :

```
5  v = prod(x)
```

× De même, comme la variable `y` contient les entiers de 1 à $2\mathtt{n}+1$, pour obtenir $(2\mathtt{n}+1)!$, on peut utiliser la commande : `prod(y)`. On obtient :

```
6  w = prod(y)
```

× Pour finir, on complète la ligne 7 de la façon suivante (toujours d'après la formule de ***5.b)***) :

```
7  u = (4^n) * v^2 / w
```

**Commentaire**

- On pouvait aussi stocker la valeur `n!` dans la variable `v` à l'aide d'une boucle for :

```
1  v = 1
2  for i = 1:n
3    v = v * i
4  end
```

- Comme dit précédemment, compléter correctement le programme **Scilab** démontre la bonne compréhension de la mécanismes en jeu et est suffisant pour obtenir les points alloués à cette question.

□

## II. Calcul du premier entier $n$ qui vérifie une condition donnée

EDHEC – 2016

- On considère une suite $(u_n)_{n\in\mathbb{N}}$ qui vérifie :

$$\forall n \in \mathbb{N},\ \mathrm{e}^{-\sqrt{u_n}} \leqslant u_n - n \leqslant \mathrm{e}^{-\sqrt{n}}$$

***a)*** Compléter les commandes **Scilab** suivantes afin qu'elles permettent d'afficher un entier $n$ pour lequel $u_n - n$ est inférieur ou égal à $10^{-4}$.

```
1  n = 0
2  while ------
3      n = ------
4  end
5  disp(n)
```

*Démonstration.*

- D'après ce qui précède :
$$\forall n \in \mathbb{N},\ u_n - n \leqslant \mathrm{e}^{-\sqrt{n}}$$
- Afin de trouver un entier $n$ tel que $u_n - n \leqslant 10^{-4}$, il suffit de trouver $N \in \mathbb{N}$ tel que :
$$\mathrm{e}^{-\sqrt{N}} \leqslant 10^{-4}$$
- On obtient alors, par transitivité :
$$u_N - N \leqslant \mathrm{e}^{-\sqrt{N}} \leqslant 10^{-4}$$
- Il s'agit donc de trouver le premier entier $N$ tel que $\mathrm{e}^{-\sqrt{N}} \leqslant 10^{-4}$.
Pour ce faire, on teste successivement tous les entiers naturels.
On arrête l'itération dès le premier entier qui satisfait cette relation.

```
2  while exp(-sqrt(n)) > 10^(-4)
3      n = n + 1
4  end
```

**Commentaire**

- La terminaison du programme présenté est assuré par le fait que $\mathrm{e}^{-\sqrt{n}} \underset{n\to+\infty}{\longrightarrow} 0$. Ainsi, pour tout $\varepsilon > 0$, il existe $N \in \mathbb{N}$ tel que : $\forall n \geqslant N$, $\mathrm{e}^{-\sqrt{n}} \leqslant \varepsilon$.
C'est notamment vrai pour $\varepsilon = 10^{-4}$.
- L'énoncé original contenait une coquille sans gravité. En effet, la quesiton posée demandait de trouver le premier $n$ tel que $v_n$ est inférieur ou égal à $10^{-4}$.
Or, la suite $(v_n)$ n'est introduite qu'en question ***4***.

□

***b)*** Le script ci-dessous affiche l'une des trois valeurs $n = 55$, $n = 70$ et $n = 85$.
Préciser laquelle en prenant $2,3$ comme valeur approchée de $\ln(10)$.

*Démonstration.*

- On cherche à déterminer l'entier $N$ précédent. Or :
$$\begin{aligned} \mathrm{e}^{-\sqrt{n}} \leqslant 10^{-4} &\Leftrightarrow -\sqrt{n} \leqslant \ln(10^{-4}) = -4\ln(10) && \text{(car } x \mapsto \ln(x) \text{ est strictement croissante sur } \mathbb{R}_+^*\text{)} \\ &\Leftrightarrow \sqrt{n} \geqslant 4\ln(10) \\ &\Leftrightarrow n \geqslant \big(4\ln(10)\big)^2 && \text{(car } x \mapsto x^2 \text{ est strictement croissante sur } [0,+\infty[\text{)} \end{aligned}$$
Ainsi, le premier entier vérifiant la relation : $\mathrm{e}^{-\sqrt{n}} \leqslant 10^{-4}$ est l'entier $N = \lceil (4\ln(10))^2 \rceil$.
*(où $x \mapsto \lceil x \rceil$ désigne la fonction partie entière par excès)*
- Cherchons maintenant une valeur approchée de $(4\ln(10))^2$.
D'après l'énoncé : $\ln(10) \simeq 2,3$, donc $4\ln(10) \simeq 9,2 \geqslant 9$.
On en déduit :
$$\big(4\ln(10)\big)^2 \geqslant 9^2 = 81$$
La seule solution possible parmi celles proposées est $n = 85$.

Le script affiche $n = 85$.

□

EML – 2016

- On considère la fonction $f$ définie par : $f(x) = \begin{cases} x^2 - x\ln(x) & \text{si } x \neq 0 \\ 0 & \text{si } x = 0 \end{cases}$

  et on définit la suite $(u_n)_{n\in\mathbb{N}^*}$ par : $\begin{cases} u_0 = \frac{1}{2} \\ \forall n \in \mathbb{N},\ u_{n+1} = f(u_n) \end{cases}$

  *(on démontre que $(u_n)$ est croissante et qu'elle converge vers 1)*

***a)*** Écrire un programme en **Scilab** qui calcule et affiche un entier naturel $N$ tel que :

$$1 - u_N < 10^{-4}$$

*Démonstration.*

```
n = 0
u = 1/2
while 1 - u >= 10^(-4)
  u = u^2 - u * log(u)
  n = n + 1
end
disp(n)
```

□

EML – 2017

- On considère la fonction $f : ]0, +\infty[ \to \mathbb{R}$ définie par $f : x \mapsto \mathrm{e}^x - \mathrm{e}\ln(x)$.
- On considère la suite réelle $(u_n)_{n\in\mathbb{N}}$ définie par : $\begin{cases} u_0 = 2 \\ \forall n \in \mathbb{N},\ u_{n+1} = f(u_n) \end{cases}$

***a)*** Écrire un programme en **Scilab** qui, étant donné un réel $A$, renvoie un entier naturel $N$ tel que $u_N \geqslant A$.

*Démonstration.*

```
A = input('A=')
N = 0
u = 2
while u < A
    u = exp(u) - %e * ln(u)
    N = N + 1
end
disp(N)
```

□

## III. Calcul de la valeur approchée de la limite d'une suite

EML – 2015

- On considère l'application $f : \mathbb{R} \to \mathbb{R},\ x \mapsto f(x) = x^3\mathrm{e}^x$.

***a)*** Montrer que la série $\sum_{n\geqslant 1} \frac{1}{f(n)}$ converge. On note $S = \sum_{n=1}^{+\infty} \frac{1}{f(n)}$.

*Démonstration.*

- La série $\sum_{n\geqslant 1} \dfrac{1}{f(n)}$ est à termes positifs.
- De plus : $\dfrac{1}{f(n)} = \dfrac{1}{n^3\,\mathrm{e}^n} = \underset{n\to+\infty}{o}\left(\dfrac{1}{n^2}\right)$. En effet, $\dfrac{\frac{1}{n^3\,\mathrm{e}^n}}{\frac{1}{n^2}} = \dfrac{n^2}{n^3\,\mathrm{e}^n} = \dfrac{1}{n\,\mathrm{e}^n} \underset{n\to+\infty}{\longrightarrow} 0$.

  Or, d'après le critère de Riemann ($2 > 1$), la série $\sum_{n\geqslant 1} \dfrac{1}{n^2}$ est convergente.

  Ainsi, d'après le théorème de négligeabilité des séries à termes positifs, la série $\sum_{n\geqslant 1} \dfrac{1}{f(n)}$ est convergente.

□

On démontre alors que : $\forall n \in \mathbb{N}^*,\ \left|S - \sum_{k=1}^{n} \dfrac{1}{f(k)}\right| \leqslant \dfrac{1}{(\mathrm{e}-1)\,\mathrm{e}^n}$.

***b)*** En déduire une fonction **Scilab** qui calcule une valeur approchée de $S$ à $10^{-4}$ près.

*Démonstration.*

- Afin de calculer une valeur approchée de $S$ à $10^{-4}$ près, il suffit de trouver $n \in \mathbb{N}^*$ tel que :

$$\frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4}$$

- En effet, d'après la question précédente, on a alors :

$$\left| S - \sum_{k=1}^{n} \frac{1}{f(k)} \right| \leqslant \frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4}$$

  et $\sum_{k=1}^{n} \dfrac{1}{f(k)}$ constitue dans ce cas l'approximation recherchée.
- On en déduit le programme suivant.

```
n = 1
S = 1 / exp(1)
while 1 / ((exp(1)-1) * exp(n)) > 10^(-4)
    n = n + 1
    S = S + 1 / (n^3 * exp(n))
end
```

**Commentaire**

Le programme précédent propose de déterminer la valeur de $n$ et de $S_n$ en procédant par itération. On peut aussi remarquer que :

$$\frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4} \Leftrightarrow \mathrm{e}^n \geqslant \frac{10^4}{(\mathrm{e}-1)} \Leftrightarrow n \geqslant 4\ln(10) - \ln(\mathrm{e}-1)$$

*(par stricte croissance de la fonction* ln*)*

Ainsi, $S_n$ est une approximation de $S$ pour tout $n \geqslant m = \lceil 4\ln(10) - \ln(\mathrm{e}-1)\rceil$.
On en déduit le programme **Scilab** suivant.

```
1  m = ceil(4 * log(10) - log(exp(1)-1))
2  S = 0
3  for i = 1:m
4      S = S + 1 / (i^3 * exp(i))
5  end
```

□

EML – 2018

- On pose : $u_0 = 4 \quad$ et $\quad \forall n \in \mathbb{N},\ u_{n+1} = \ln(u_n) + 2$.

On devait démontrer les propriétés suivantes.

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

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

*Démonstration.*

```
1  function u = suite(n)
2      u = 4
3      for k = 1:n
4          u = log(u) + 2
5      end
6  endfunction
```

Expliquons un peu ce programme.
La variable `u` est créée pour contenir successivement les valeurs $u_0$, $u_1$, …, $u_n$.

- On initialise donc cette variable à $u_0 = 4$ avec la ligne 2

```
2      u = 4
```

- On met ensuite à jour `u` de manière itérative avec la ligne 4

```
4          u = log(u) + 2
```

**Commentaire**

- On décrit ici de manière précise les instructions afin d'aider le lecteur un peu moins habile en **Scilab**.
  Cependant, l'écriture du script démontre la compréhension de toutes les commandes en question et permet sans doute d'obtenir la totalité des points alloués à cette question.
- Si on avait souhaité afficher tous les $n$ premiers termes de la suite $(u_n)$, on aurait modifié le script précédent de la façon suivante :

```
1  function u = suite(n)
2      u = zeros(1, n)
3      u(1) = 4
4      for k = 2:n
5          u(k) = log(u(k-1)) + 2
6      end
7  endfunction
```

□

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

```
1  function b = valeur_approchee(epsilon)
2      n = 0
3      while ...........
4          n = n + 1
5      end
6      b = suite(n)
7  endfunction
```

*Démonstration.*

- D'après la question ***6.b)*** :

$$\forall n \in \mathbb{N},\ 0 \leqslant u_n - b \leqslant \frac{1}{2^{n-1}}$$

  S'il existe $N \in \mathbb{N}$ tel que $\dfrac{1}{2^{n-1}} \leqslant \varepsilon$, on obtiendra par transitivité :

$$0 \leqslant u_N - b \leqslant \varepsilon$$

  Donc $u_N$ est une valeur approchée de $b$ à $\varepsilon$ près.

- On complète alors le programme **Scilab** de la façon suivante :

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

□

EML – 2019

- On considère la fonction $f$ définie sur $]0, +\infty[$ par :

$$\forall t \in ]0, +\infty[,\ f(t) = t + \frac{1}{t}$$

- On introduit la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par :

$$u_1 = 1 \quad \text{et} \quad \forall n \in \mathbb{N}^*,\ u_{n+1} = u_n + \frac{1}{n^2\, u_n} = \frac{1}{n}\, f(n\, u_n)$$

- Dans l'exercice, on devait démontrer la convergence de la suite $(u_n)$ vers un réel $\ell$ (qu'il ne fallait pas déterminer). De plus, on devait démontrer :

$$\forall p \geqslant 2,\ 0 \leqslant \ell - u_p \leqslant \frac{1}{p-1} \quad (*)$$

***a)*** Recopier et compléter les lignes 3 et 4 de la fonction **Scilab** suivante afin que, prenant en argument un entier $n$ de $\mathbb{N}^*$, elle renvoie la valeur de $u_n$.

```
1  function u=suite(n)
2    u = 1
3    for k = ..................
4      u = ..................
5    end
6  endfunction
```

*Démonstration.*

```
1  function u=suite(n)
2    u = 1
3    for k = 1:(n-1)
4      u = (1/n) * (n*u + 1/(n*u))
5    end
6  endfunction
```

Détaillons l'obtention de ce programme.
La variable u est crée pour contenir successivement les valeurs $u_1$, …, $u_n$.

- On initialise donc cette variable à $u_1 = 1$ avec la ligne 2.

```
2    u = 1
```

- On met ensuite à jour u à l'aide d'une structure itérative (boucle for) avec les lignes 3 à 5.

```
3    for k = 1:(n-1)
4      u = (1/n) * (n*u + 1/(n*u))
5    end
```

**Commentaire**

- On décrit ici de manière précise les instructions afin d'aider le lecteur un peu moins habile en **Scilab**. Cependant, l'écriture du script démontre la compréhension de toutes les commandes en question et permet sans doute d'obtenir la totalité des points alloués à cette question.
- On pouvait également coder la fonction $f$ dans un script à part. On aurait alors obtenu les deux programmes suivants :

```
function y=f(t)
  y = t + 1/t
endfunction
```

```
function u=suite(n)
  u = 1
  for k = 1:(n-1)
    u = (1/n) * f(n*u)
  end
endfunction
```

□

***b)*** Déduire de la propriété $(*)$ une fonction **Scilab** qui renvoie une valeur approchée de $\ell$ à $10^{-4}$ près.

*Démonstration.*

- On cherche ici à trouver un entier $N$ tel que $u_N$ est une valeur approchée de $\ell$ 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 :

```
function l = valeur_approchee()
  n = 2
  while 1 / (n-1) > 10^(-4)
    n = n + 1
  end
  l = suite(n)
endfunction
```

Détaillons les éléments de ce script.

**- Début du script**

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

```
  n = 2
```

- **Structure itérative**

Les lignes 3 à 5 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 $\ell$ : 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 $\ell$. 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.

□

**ECRICOME – 2018**

- Pour tout entier naturel $n$ non nul, on pose : $u_n = \displaystyle\sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.
- On démontrait que la suite $(u_n)$ était convergente, vers une limite notée $\gamma \in \mathbb{R}$ puis :
$$\forall n \in \mathbb{N}^*,\ |u_n - \gamma| \leqslant \frac{1}{n}$$

***a)*** Écrire une fonction d'en-tête : `function y = u(n)` qui prend en argument un entier naturel $n$ non nul et qui renvoie la valeur de $u_n$.

*Démonstration.*

```
1  function y = u(n)
2      S = 0
3      for k = 1:n
4          S = S + 1/k
5      end
6      y = S - log(n)
7  endfunction
```

Détaillons les différents éléments de ce code :

× en ligne 2, on crée la variable `S` dont le but est de contenir, en fin de programme $\sum_{k=1}^{n} \frac{1}{k}$.
Cette variable `S` est donc initialisée à 0.

× de la ligne 3 à la ligne 5, on met à jour la variable `S` à l'aide d'une boucle.
Pour ce faire, on ajoute au $k^{\text{ème}}$ tour de boucle la quantité $\frac{1}{k}$.
Ainsi, `S` contient bien $\sum_{k=1}^{n} \frac{1}{k}$ en sortie de boucle.

× en ligne 6, on affecte à la variable `y` la valeur $u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.

**Commentaire**

Pour le calcul de la somme $\sum_{k=1}^{n} \frac{1}{k}$, on peut aussi tirer profit des fonctionnalités **Scilab** :

```
S = sum(1 ./ 1:n)
```

Pour bien comprendre cette instruction, rappelons que :

× l'instruction `1:n` permet de créer la matrice ligne $(1 \quad 2 \quad \dots \quad n)$.

× l'opérateur `./` permet d'effectuer la division terme à terme.
Ainsi, l'instruction `1 ./ 1:n` permet de créer la matrice ligne $(\frac{1}{1} \quad \frac{1}{2} \quad \dots \quad \frac{1}{n})$.

× la fonction `sum` permet de sommer tous les coefficients d'une matrice.

On obtient donc bien la somme à calculer par cette méthode.

□

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

```
1  eps = input('Entrer un réel strictement positif : ')
2  n = floor(1/eps) + 1
3  disp(u(n))
```

*Démonstration.*

- Ce script a pour but d'afficher une valeur approchée de $\gamma$ à $\varepsilon$ près (où $\varepsilon$ 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 $\gamma$ à $\varepsilon$ près. □

## IV. Calcul de la valeur approchée des éléments d'une suite

ECRICOME – 2019

- Pour tout entier $n$ non nul, on note $h_n$ la fonction définie sur $\mathbb{R}_+^*$ par :
$$\forall x > 0,\quad h_n(x) = x^n + 1 + \frac{1}{x^n}$$

- On devait démontrer 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 : $0 < u_n < 1 < v_n$.

***a)*** Écrire une fonction **Scilab** d'en-tête `function y = 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.

*Démonstration.*

```
1 function y = h(n, x)
2   y = x^n + 1 + (1 / x^n )
3 endfunction
```

**Commentaire**

Il n'y a aucune difficulté à coder en **Scilab** une fonction dont l'expression est donnée dans l'énoncé. Il est donc impensable de ne pas traiter cette question. □

***b)*** 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 :

```
function res=v(n)
  a = 1
  b = 3
  while (b-a) > 10^(-5)
    c = (a+b)/2
    if h(n,c) < 4 then
      ........
    else
      ........
    end
  end
  ..........
endfunction
```

*Démonstration.*
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 $\varepsilon$.

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

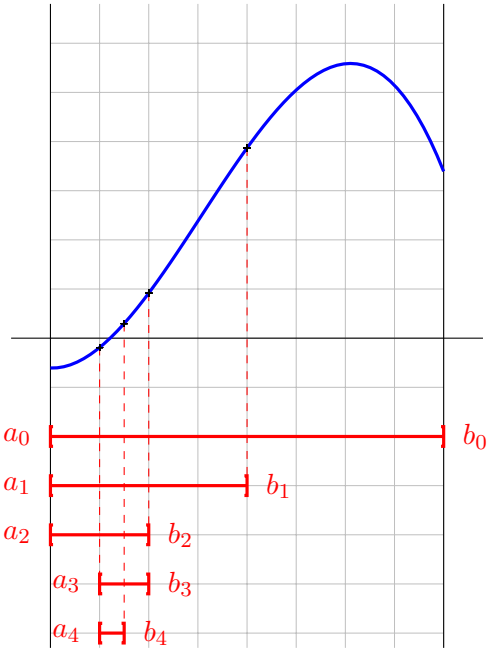
Supposons : $f(a)\ f(b) \leqslant 0$.

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.
- Il reste enfin à adapter cet algorithme à l'énoncé.
  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 then
7          a = c
```

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

```
8         else
9           b = c
10        end
```

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.

```
12      res = a
```

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

```
12      res = b
```

Ou encore le point milieu :

```
12      res = (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 :

$$\begin{aligned}
\frac{2}{2^m} \leqslant 10^{-5} &\Leftrightarrow \frac{2^m}{2} \geqslant 10^5 && \textit{(par stricte croissance de la fonction inverse sur } \mathbb{R}_+^*\textit{)}\\
&\Leftrightarrow 2^m \geqslant 2 \times 10^5 && \textit{(car } 4 > 0\textit{)}\\
&\Leftrightarrow m\ \ln(2) \geqslant \ln(2) + 5\ \ln(10) && \textit{(par stricte croissance de la fonction } \ln \textit{ sur } \mathbb{R}_+^*\textit{)}
\end{aligned}$$

  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 **Scilab** (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.

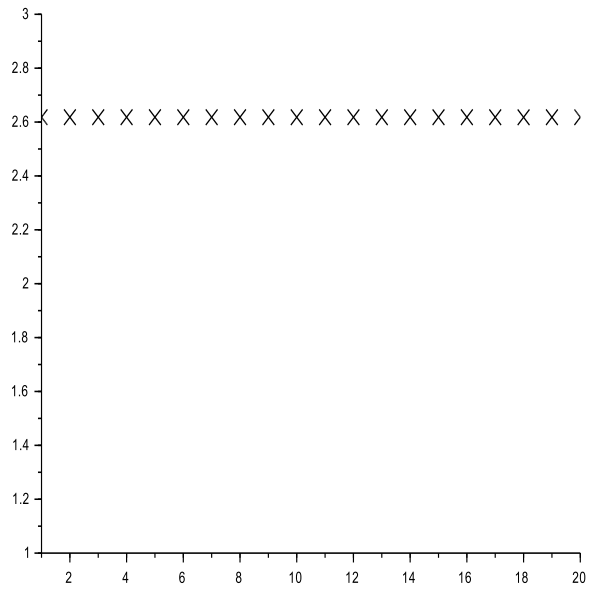
```
function res = v(n)
  a = 1
  b = 3
  while (b-a) > 10^(-5)
    c = (a+b) / 2
    if h(n,c) < 4 then
      a = c
    else
      b = c
    end
  end
  res = a
endfunction
```

□

***c)*** À la suite de la fonction `v`, on écrit le code suivant :

```
X = 1:20
Y = zeros(1,20)
for k = 1:20
  Y(k) = v(k)^k
end
plot2d(X, Y, style=-2, rect=[1,1,20,3])
```

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 ?

*Démonstration.*

- Le programme commence par définir deux tableaux (matrices lignes) `X` et `Y`.

```
1  X = 1:20
2  Y = zeros(1,20)
```

Le tableau `X` contient initialement $[1, 2, \ldots, 20]$, c'est-à-dire les 20 premiers entiers non nuls.
Le tableau `Y` est destiné à contenir les 20 premières valeurs de la suite $({v_n}^n)$.
Il est initialement rempli de 0.

- À l'aide d'une boucle, la $k^{\text{ème}}$ case `Y` est modifiée de sorte à contenir une valeur approchée de $v_k^k$.

```
3  for k = 1:20
4    Y(k) = v(k) ^ k
5  end
```

- 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.

```
6  plot2d(X, Y, style=-2, rect=[1,1,20,3])
```

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. □