Approximations de nombres

Auteur : Raymond Moché

Enseignants (lycée & université)

Table des matières

Objet du papier : Variations algorithmiques autour du thème « tout nombre réel est limite d'une suite de nombres rationnels » : approximations de nombres réels $\geqslant 0$ par des sommes de nombres de la forme $\frac{1}{n}$, $\frac{1}{n^2}$, $\sin\left(\frac{\pi}{2n}\right)$, *etc*, par exemple :

$$\pi \approx 3+\frac{1}{8}+\frac{1}{61}+\frac{1}{5020}+\frac{1}{128541454}=\frac{123659257071119}{39361964043880} \quad \text{à } 10^{-9} \text{ près}$$

$$e \approx 2+2\cdot\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\frac{1}{5^2}+\frac{1}{15^2}+\frac{1}{67^2}+\frac{1}{535^2}+\frac{1}{8986^2}=\frac{10152826873311159749}{3735016276465760400} \quad \text{à } 10^{-11} \text{ près}$$

$$\pi \approx 3+\sin\left(\frac{\pi}{2\cdot 12}\right)+\sin\left(\frac{\pi}{2\cdot 142}\right)+\sin\left(\frac{\pi}{2\cdot 331322}\right)+\sin\left(\frac{\pi}{2\cdot 141834644451}\right) \quad \text{à } 10^{-22} \text{ près}$$

Mots-clefs : Algorithmique : « Xcas ». **Analyse :** approximation de nombres réels par des nombres spéciaux, approximation par défaut, erreur d'approximation, fonction inversible, fonction réciproque.

Remarques :

- ✓ Nous disposions d'un algorithme « Xcas » écrit sans trop réfléchir[1] qui donne des valeurs approchées par défaut aussi précises que l'on veut de la partie décimale de tout nombre réel $x \geqslant 0$ sous forme de sommes de termes en $\frac{1}{n}$, $n \in \mathbb{N}$. Cet algorithme est proposé en tant que curiosité pour la TS Spé Math (cf. [1]).
- ✓ En justifiant cet algorithme, nous avons été amenés à élargir le cadre de l'étude, ce qui a conduit aux théorèmes (2.1) et (2.2) et au corollaire (2.3). Le premier théorème est un théorème de convergence, le second un théorème d'approximation

1. parce qu'il s'agit d'une idée naturelle

qui justifie la plupart des algorithmes qui suivent, le corollaire justifiant un cas particulier.

- ✓ Nous avons utilisé « Xcas » de préférence à « scilab » par exemple parce qu'il n'exécute les calculs numériques que quand on lui demande (calcul formel puis calcul numérique).
- ✓ Nous avons ignoré les erreurs d'arrondi et d'approximation auxquelles les calculs numériques donnent lieu.
- ✓ Une partie du code LaTeX a été générée directement par « Xcas ».
- ✓ Pour nous, ce papier est une curiosité amusante. Nous ne voyons pas bien à quoi ça peut servir.

1 Définition du problème traité

1.1 Introduction : développement décimal d'un nombre

Soit x un nombre réel $\geqslant 0$. On sait que si on pose [2]

$$\forall n \in \mathbb{N}, \quad x_n = \frac{1}{10^n} \cdot \left[10^n x\right]$$

ces nombres vérifient les inégalités

$$\forall n \geqslant 0, \quad x_n \leqslant x < x_n + \frac{1}{10^n} \quad \text{et} \quad x = \lim_{n \to +\infty} x_n$$

Il est clair que

- ✓ si x n'est pas un nombre décimal, x_n est le nombre réduit à la partie entière de x et à ses n premières décimales ;
- ✓ si x est un nombre décimal, x_n a la même signification pourvu que le développement décimal de x choisi soit celui qui ne comporte que des zéros à partir d'un certain rang.

Si on considère x_n comme une valeur approchée de x, c'est une approximation de x par défaut, l'erreur d'approximation $d = x - x_n$ vérifiant

$$0 \leqslant d < \frac{1}{10^n}$$

Cette suite d'approximation par défaut est évidemment croissante.

Exemple 1.1 Si $x = \pi$, les 6 premiers termes de la suite $(x_n,\, n \geqslant 0)$ sont

$$x_0 = 3,\ x_1 = 3.1,\ x_2 = 3.14,\ x_3 = 3.141,\ x_4 = 3.1415,\ x_5 = 3.14159$$

ce que l'on peut écrire

$$x_0 = 3 \quad x_1 = 3 + \frac{1}{10} \quad x_2 = 3 + \frac{1}{10} + \frac{4}{10^2} \qquad x_3 = 3 + \frac{1}{10} + \frac{4}{10^2} + \frac{1}{10^3}$$

$$x_4 = 3 + \frac{1}{10} + \frac{4}{10^2} + \frac{1}{10^3} + \frac{5}{10^4} \quad x_5 = 3 + \frac{1}{10} + \frac{4}{10^2} + \frac{1}{10^3} + \frac{5}{10^4} + \frac{9}{10^9} \ \triangle$$

2. Dans tout ce papier, [ ] désigne la fonction partie entière.

Intuitivement, il est évident - comme le montre cet exemple - que l'on peut approcher π par défaut et de manière aussi précise que l'on veut par des nombres

$$3+\sum_{j=1}^{k}\frac{1}{10^{n_j}} \quad \text{ou} \quad 3+\sum_{j=1}^{k}\varphi\Big(\frac{1}{n_j}\Big)$$

φ désignant la fonction $x \mapsto 10^{-\frac{1}{x}}$, définie sur l'intervalle $]0,1]$, certains des exposants $n_1, \ldots, n_k$ (entiers > 0) pouvant être égaux.

1.2 Position du problème

Nous allons ci-dessous nous intéresser à un type d'approximation plus général dont nous fixons maintenant le cadre :

Soit f une fonction continue strictement croissante définie sur l'intervalle $]0,1]$ telle que

$$f(x) \xrightarrow[x\to 0+]{} 0 \quad \text{et} \quad f(1)=1 \tag{1}$$

On sait que f est une bijection de $]0,1]$ sur lui-même, que l'application réciproque g [3] est continue, strictement croissante et vérifie

$$g(x) \xrightarrow[x\to 0+]{} 0 \quad \text{et} \quad g(1)=1$$

Dans la suite, f désignera une telle fonction.

Exemples de fonctions f utilisées dans ce papier [4] **:**

Tableau 1.2

$f_1(x)=x$	$g_1(x)=x$	$f_2(x)=x^2$	$g_2(x)=\dfrac{1}{\sqrt{x}}$
$f_3(x)=sin\Big(\dfrac{\pi x}{2}\Big)$	$g_3(x)=\dfrac{2}{\pi}\cdot\arcsin(x)$	$f_4(x)=10^{\frac{x-1}{x}}$	$g_4(x)=\dfrac{1}{1-\lg(x)}$
$f_5(x)=e^{\frac{x-1}{x}}$	$g_5(x)=\dfrac{1}{1-\ln(x)}$		

- ✓ Ces exemples sont simples parce que la fonction réciproque g est calculable, ce qui n'est pas toujours le cas.
- ✓ Il y a évidemment une infinité de fonctions continues strictement croissantes définies sur $]0,1]$ qui vérifient (1), par exemple les fonctions $x \mapsto x^\alpha$, $\alpha \in \mathbb{R}^{+*}$.
- ✓ En multipliant ou en composant des fonctions f ou g du tableau (1.2), on peut aussi en définir une infinité.

3. habituellement notée g
4. lg désigne la fonction logarithme à base 10.

Énoncé du problème :

Étant donné des réels $x \geqslant 0$ et $E > 0$ et une fonction f du type décrit ci-dessus, trouver une valeur approchée a de x

✓ de la forme

$$a = [x] + \sum_{j=1}^{k} f\Big(\frac{1}{n_j}\Big)$$

où si $k \geqslant 1$, $n_1, \ldots, n_k$ sont des entiers tels que $1 \leqslant n_1 \leqslant n_2 \leqslant \cdots \leqslant n_k$ tandis que si $k = 0$, la suite $n_1, \ldots, n_k$ est vide et $a = [x]$ [a]

✓ telle que

$$0 \leqslant d = x - a < E$$

a. en vertu de la convention que la somme de la famille vide de nombres réels vaut 0

2 Solution du problème

Dans la suite, on appelle $\mathcal{R}$ l'ensemble des nombres réels $\geqslant 0$ de la forme

$$n + \sum_{j=1}^{k} f\Big(\frac{1}{n_j}\Big), \quad k \in \mathbb{N}, \quad n, n_1, \ldots, n_k \in \mathbb{N}, \quad 1 \leqslant n_1 \leqslant \cdots \leqslant n_k$$

Théorème 2.1 *Pour tout nombre réel $x \geqslant 0$ n'appartenant pas à $\mathcal{R}$, il existe une suite croissante* [a] *$(n_k,\ k \geqslant 1)$ d'entiers $\geqslant 2$ telle que*

$$n_k \xrightarrow[k\to+\infty]{} +\infty \qquad \text{et} \qquad x = [x] + \sum_{k=1}^{+\infty} f\Big(\frac{1}{n_k}\Big). \quad \blacktriangle \tag{2}$$

a. « croissante » signifie « croissante au sens large ».

Démonstration : Posons $x_0 = [x]$. Dans le cas considéré,

$$x_0 = [x] \iff x_0 \in \mathbb{N} \quad \text{et} \quad 0 < x - x_0 < 1$$

(parce que l'égalité $x = x_0$ est impossible, x n'étant pas entier, puisque $x \notin \mathcal{R}$). Comme

$$]0, 1[= \bigcup_{n=2}^{+\infty} \Big[f\Big(\frac{1}{n}\Big), f\Big(\frac{1}{n-1}\Big)\Big[$$

les intervalles de cette réunion étant deux à deux disjoints, $x - x_0$ appartient à un seul d'entre eux noté $\Big[f\Big(\frac{1}{n_1}\Big), f\Big(\frac{1}{n_1-1}\Big)\Big[$, où $n_1 \geqslant 2$. En fait, $x - x_0 \in \Big]f\Big(\frac{1}{n_1}\Big), f\Big(\frac{1}{n_1-1}\Big)\Big[$ car l'égalité $x = x_0 + f\Big(\frac{1}{n_1}\Big)$ est impossible, puisque $x \notin \mathcal{R}$ [5].

En vue du calcul de n_1, notons que

$$x - x_0 \in \Big[f\Big(\frac{1}{n_1}\Big), f\Big(\frac{1}{n_1-1}\Big)\Big[\iff n_1 = \min\,(m,\ m \in \mathrm{N}^* \text{ et } f\Big(\frac{1}{m}\Big) \leqslant x - x_0)$$

5. Cet argument sera encore utilisé sans le dire dans les calculs qui suivent.

ce qui promet un calcul lent avec beaucoup de tests ou, en utilisant la fonction g,

$$x-x_0\in\Big[f\Big(\frac{1}{n_1}\Big),f\Big(\frac{1}{n_1-1}\Big)\Big[\iff f\Big(\frac{1}{n_1}\Big)\leqslant x-x_0<f\Big(\frac{1}{n_1-1}\Big)\iff n_1=-\Big[\frac{-1}{g(x-x_0)}\Big]$$

Si on pose $x_1=x_0+f\Big(\dfrac{1}{n_1}\Big)$, nous avons donc

$$0\leqslant x_0<x_1<x<x_1+f\Big(\frac{1}{n_1-1}\Big)-f\Big(\frac{1}{n_1}\Big)<x_1+1$$

n_1 et x_1 étant définis, on peut définir par récurrence une suite d'entiers $(n_k,\ k\geq 1)$ et une suite de réels $(x_k,\ k\geq 1)$ qui ont les propriétés suivantes :

$$\forall k\geqslant 1,\quad n_k=\min\,(m,\ m\in\mathrm{N}^*\text{ et }f\Big(\frac{1}{m}\Big)\leqslant x-x_{k-1})=-\Big[\frac{-1}{g(x-x_{k-1}}\Big] \tag{3}$$

$$\forall k\geqslant 1,\quad x_k=x_{k-1}+f\Big(\frac{1}{n_k}\Big)=x_0+\sum_{j=1}^{k}f\Big(\frac{1}{n_j}\Big) \tag{4}$$

$$\forall k\geqslant 1,\quad 0\leqslant x_0<x_1<\ldots<x_k<x<x_k+f\Big(\frac{1}{n_k-1}\Big)-f\Big(\frac{1}{n_k}\Big)<x_k+1 \tag{5}$$

$$\forall k\geqslant 1,\quad 2\leqslant n_1\leqslant n_2\leqslant\cdots\leqslant n_k \tag{6}$$

En effet, supposons que l'on ait pu définir ainsi les nombres $n_1,\ldots,n_k$ et $x_1,\ldots,x_k$. Comme $0<x-x_k<1$, on peut poser

$$n_{k+1}=-\Big[\frac{-1}{g(x-x_k)}\Big]=\min\,(m,\ m\in\mathrm{N}^*\text{ et }f\Big(\frac{1}{m}\Big)\leqslant x-x_k) \tag{7}$$

ce qui implique $n_{k+1}\geqslant n_k$, car $x_{k-1}<x_k$. Posons

$$x_{k+1}=x_k+f\Big(\frac{1}{n_{k+1}}\Big) \tag{8}$$

Cela permet de traduire

$$f\Big(\frac{1}{n_{k+1}}\Big)<x-x_k<f\Big(\frac{1}{n_{k+1}-1}\Big),$$

cf. (7), par

$$x_{k+1}<x<x_{k+1}+f\Big(\frac{1}{n_{k+1}-1}\Big)-f\Big(\frac{1}{n_{k+1}}\Big)<x_{k+1}+1 \tag{9}$$

Cela montre que l'on peut effectivement définir par récurrence les suites $(n_k,\ k\geq 1)$ et $(x_k,\ k\geq 1)$ ayant les propriétés (3), (4), (5) et (6).

Démontrons maintenant par l'absurde que

$$n_k\xrightarrow[k\to+\infty]{}+\infty \tag{10}$$

Si cette propriété était fausse, comme la suite $(n_k,\ k\geq 1)$ est croissante, elle serait constante à partir d'un certain rang. D'après (4), la suite $(x_k,\ k\geq 1)$ tendrait donc vers $+\infty$, ce qui est faux puisqu'elle est majorée par x.

On déduit de (10), d'après (1) et (5) , que

$$x_k \xrightarrow[k\to+\infty]{} x$$

ce qui prouve que $x = [x] + \sum_{k=1}^{+\infty} f\Big(\frac{1}{n_k}\Big)$ et termine la démonstration du théorème 2.1. ▼

Comme cette démonstration est constructive (récurrence), elle pourra se traduire facilement par un algorithme. Compte tenu de la définition de $\mathcal{R}$, on peut donc énoncer

Théorème 2.2 *Pour tous réels $x \geqslant 0$ et $E > 0$, il existe deux suites finies éventuellement vides d'entiers, notées $M = (m_1, \ldots, m_k)$ et $C = (c_1, \ldots, c_k)$ et un entier $n \geqslant 0$ tels que*

$$2 \leqslant m_1 < m_2 < \cdots < m_k, \quad 1 \leqslant c_1, \ldots, c_k \tag{11}$$

et que

$$a = n + \sum_{j=1}^{k} c_j f\Big(\frac{1}{m_j}\Big)$$

étant considéré comme une valeur approchée de x, l'erreur d'approximation d vérifie

$$0 \leqslant d = x - a < E. \quad ▲$$

Démonstration Ce résultat est évident d'après le théorème précédent et la définition de $\mathcal{R}$. Si $x \notin \mathcal{R}$, on prend dans la série (2) suffisamment de termes pour que l'erreur d'approximation par défaut soit strictement majorée par E puis, que x appartienne ou non à $\mathcal{R}$, on regroupe les termes égaux de la forme $f\big(\frac{1}{n}\big)$ et on note $c_1, \ldots, c_k$ les effectifs des regroupements pour obtenir (13).

La preuve qui suit, inutile, regroupe les cas $x \in \mathcal{R}$ et $x \notin \mathcal{R}$. C'est pratiquement un algorithme.

Deux réels quelconques $x \geqslant 0$ et $E > 0$ (et une fonction f) sont donc donnés.

Posons

$$\begin{cases} x_0 = [x] & (\Longrightarrow x_0 \leqslant x < x_0 + 1) \\ d_0 = x - x_0 & (\Longrightarrow 0 \leqslant d_0 < 1) \end{cases}$$

Si $d_0 < E$, on pose

$$\begin{cases} n = x_0 \\ M = C = \emptyset \end{cases}$$

Sinon, $d_0 \geqslant E \quad (\Longrightarrow x_0 < x)$, on pose

$$\begin{cases} m_1 = \min\Big(m,\, m \in \mathbb{N}^*,\, x_0 + f\Big(\frac{1}{m}\Big) \leqslant x\Big) = -\Big[\frac{-1}{g(d_0)}\Big] & (\Longrightarrow m_1 \geqslant 2) \\ c_1 = \max\Big(c,\, c \in \mathbb{N}^*,\, x_0 + cf\Big(\frac{1}{m_1}\Big) \leqslant x\Big) = \Big[\dfrac{x - x_0}{f\Big(\frac{1}{m_1}\Big)}\Big] & \\ x_1 = x_0 + c_1 f\Big(\frac{1}{m_1}\Big) & (\Longrightarrow x_1 \leqslant x < x_1 + f\Big(\frac{1}{m_1}\Big)) \\ d_1 = x - x_1 & \end{cases}$$

Si $d_1 < E$, on pose

$$\begin{cases} n = x_0 \\ M = (m_1) \\ C = (c_1) \end{cases}$$

Sinon, nous avons

$$\begin{cases} E \leqslant d_1 \\ x_1 < x_1 + E \leqslant x < x_1 + f\Big(\dfrac{1}{m_1}\Big) \\ E < f\Big(\dfrac{1}{m_1}\Big) \iff m_1 < \dfrac{1}{g(E)} \end{cases}$$

Dans ce cas, on pose

$$\begin{cases} m_2 = \min\Big(m,\, m \in \mathbb{N}^*,\, x_1 + f\Big(\dfrac{1}{m}\Big) \leqslant x\Big) = -\Big[\dfrac{-1}{g(d_1)}\Big] \ (\Longrightarrow m_1 < m_2) \\ c_2 = \max\Big(c,\, c \in \mathbb{N}^*,\, x_1 + cf\Big(\dfrac{1}{m_2}\Big) \leqslant x\Big) = \Big[\dfrac{x - x_1}{f\Big(\dfrac{1}{m_2}\Big)}\Big] \\ x_2 = x_1 + c_2 f\Big(\dfrac{1}{m_2}\Big) \qquad (\Longrightarrow x_2 \leqslant x < x_2 + f\Big(\dfrac{1}{m_2}\Big)) \\ d_2 = x - x_2 \end{cases}$$

Si $d_2 < E$, nous arrêtons le calcul comme précédemment. Sinon,

$$\begin{cases} E \leqslant d_2 \\ x_2 < x_2 + E \leqslant x < x_2 + f\Big(\dfrac{1}{m_2}\Big) \\ E < f\Big(\dfrac{1}{m_2}\Big) \iff m_2 < \dfrac{1}{g(E)} \end{cases}$$

On continuera le calcul tant que l'on n'aura pas obtenu une erreur d'approximation strictement inférieure à E, ce qui arrivera nécessairement, car si pour un certain entier $p \geqslant 1$, on a pu définir suivant le procédé indiqué ci-dessus les suites $(m_1, \ldots, m_p)$ et $(c_1, \ldots, c_p)$, les inégalités

$$m_1 < m_2 < \cdots < m_p < \frac{1}{g(E)} \Longrightarrow p + 1 < \frac{1}{g(E)}$$

sont satisfaites, ce qui montre que $p \leqslant \Big[\dfrac{1}{g(E)}\Big] - 1$. ▼

Remarque : Ce théorème ne dit pas que les entiers $n, m_1, \ldots, m_k, c_1, \ldots, c_k$ sont uniques, ce serait faux. Par exemple, si $f(x) = x$, $x = 1$ et $E = 10^{-1}$,

$$n = 1,\ M = C = \emptyset \quad \text{et} \quad n = 0,\ M = (1),\ C = (1)$$

sont des approximations exactes de x.

Nous terminons par un cas particulier intéressant où il sera inutile de rechercher C.

Corollaire 2.3 *Si la fonction f vérifie*

$$\forall n \geqslant 2, \quad 2f\Big(\frac{1}{n}\Big) \geqslant f\Big(\frac{1}{n-1}\Big) \tag{12}$$

pour tous réels $x \geqslant 0$ et $E > 0$, il existe une suite finie éventuellement vide d'entiers notée $M = (m_1, \ldots, m_k)$ et un entier $n \geqslant 0$ tels que

$$2 \leqslant m_1 < m_2 < \cdots < m_k \tag{13}$$

et que

$$a = n + \sum_{j=1}^{k} f\Big(\frac{1}{m_j}\Big)$$

étant considéré comme une valeur approchée de x, l'erreur d'approximation d vérifie

$$0 \leqslant d = x - a < E. \quad \blacktriangle$$

Démonstration : Le problème est de démontrer que les coefficients $c_1, \ldots, c_k$ définis dans le théorème précédent sont tous égaux à 1 (si $k \geqslant 1$). Or, pour tout entier j, $1 \leqslant j \leqslant k$, par définition de m_j et d'après (12),

$$x_{j-1} + f\Big(\frac{1}{m_j}\Big) \leqslant x < x_{j-1} + f\Big(\frac{1}{m_j - 1}\Big) \leqslant x_{j-1} + 2f\Big(\frac{1}{m_j}\Big)$$

ce qui prouve que $c_j = 1$. ▼

La propriété (12) est vérifiée par la fonction $f(x) = x$. Cela donnera dans ce cas des algorithmes simplifiés.

3 Applications algorithmiques

3.1 Algorithmes génériques

Les algorithmes suivants sortent le quadruplet (L, C, a, d) ou le triplet (L, a, d) (lorsque C est inutile) où

$$L = (n, m_1, \ldots, m_k), \quad C = (c_1, \ldots, c_k)$$

a étant l'approximation par défaut désirée, d l'erreur d'approximation réelle ($< E$) aux problèmes de calcul près.

Pour distinguer ces algorithmes, on dit qu'un algorithme est « lent » si les $m_1, \ldots, m_k$ sont calculés comme des minima, « rapide » s'ils sont calculés à l'aide de la fonction réciproque g.

Algorithme naturel : *Approximation lente par des sommes en f(1/n)*

Entrées : x, E ;
Sorties : L, C, a, d ;
Début du traitement des données :

```
Lire : x, E ;
Poser a = [x] ;
Poser d = x − a ;
Poser m = 1 ;
Poser L = [a] ;
Poser C = [ ] ;
Tant que d >= E
    poser m=m+1 ;
    Si a + f(1/m) ⩽ x alors
        poser c = 0 ;
        Tant que a + f(1/m) ⩽ x
            poser a = a + f(1/m) ;
            poser c = c + 1 ;
        Fin de la boucle tant que
        poser d = x − a ;
        poser L = [L, m] ;
        poser C = [C, c] ;
    Fin de l'instruction conditionnelle
Fin de la boucle tant que
Afficher L, C, a, d ;
```

Fin du traitement des données

Algorithme 3.1

Voici l'algorithme rapide correspondant :

Algorithme naturel : *Approximation rapide par des sommes en f(1/n)*

Entrées : x, E ;
Sorties : L, C, a, d ;
Début du traitement des données :

Lire : x, E ;
Poser $a = [x]$;
Poser $d = x - a$;
Poser $L = [a]$;
Poser $C = [\,]$;
Tant que $d >= E$

poser $m = -\left[\dfrac{-1}{g(d)}\right]$;
poser $c = \left[\dfrac{d}{f(1/m)}\right]$;
poser $a = a + cf(1/m)$;
poser $d = x - a$;
poser $L = [L, m]$;
poser $C = [C, c]$;

Fin de la boucle tant que
Afficher L, C, a, d ;

Fin du traitement des données

Algorithme 3.2

3.2 Algorithmes dans le cas $f(x) = x$

Dans ce cas, d'après le corollaire (2.3), il est inutile de calculer C qui est vide ou ne comprend que des 1. On peut donc simplifier les algorithmes précédents.

Algorithme naturel : *Approximation lente par des sommes en 1/n*

Entrées : x, E ;
Sorties : L, a, d ;
Début du traitement des données :

- *Lire :* x, E ;
- *Poser* $a = [x]$;
- *Poser* $d = x - a$;
- *Poser* $m = 1$;
- *Poser* $L = [a]$;
- **Tant que** $d >= E$
  - *Poser* $m=m+1$;
  - **Si** $a + 1/m \leqslant x$ **alors**
    - *poser* $a = a + 1/m$;
    - *poser* $d = x - a$;
    - *poser* $L = [L, m]$;
  - **Fin de l'instruction conditionnelle**
- **Fin de la boucle tant que**
- **Afficher** L, a, d ;

Fin du traitement des données

Algorithme 3.3

Voici l'algorithme rapide correspondant :

Algorithme naturel : *Approximation rapide par des sommes en 1/n*

Entrées : x $(\geqslant 0)$, E (> 0) ;
Sorties : L, a, d ;
Début du traitement des données :

- *Lire :* x, E ;
- *Poser* $a = [x]$;
- *Poser* $d = x - a$;
- *Poser* $L = [a]$;
- **Tant que** $d >= E$
  - *poser* $m = -\Big[-\Big(g(d)\Big)^{-1}\Big]$;
  - *poser* $a = a + f\Big(1/m\Big)$;
  - *poser* $d = x - a$;
  - *poser* $L = [L, m]$;
- **Fin de la boucle tant que**
- **Afficher** L, a, d ;

Fin du traitement des données

Algorithme 3.4

L'algorithme (3.3) (l'algorithme lent) peut se traduire ainsi lorsqu'on utilise « Xcas » :
1

```
Approx(x,E):={
```

```
local a,m,d,L;
a:=floor(x);
d:=x-a;
m:=1;
L:=[a];
while(d>=E)
{m:=m+1;
if(a+1/m<=x)
{a:=a+1/m;
d:=x-a;
L:=append(L,m);
};
};
d:=evalf(d);
return L,a,d;
}
:;
```

// Parsing Approx
// Success compiling Approx

$$\text{Done} \tag{14}$$

2 `Approx(2/3,10^-1) ;`

$$[0,2,6], \frac{2}{3}, 0.00000000000000000 \tag{15}$$

3 `Approx(pi,10^-3) ;`

$$[3,8,61], \frac{1533}{488}, 0.19921096684227813e-3 \tag{16}$$

4 `Approx(pi,10^-6) ;`
Evaluation time : 1.05

$$[3,8,61,5020], \frac{1924037}{612440}, 0.77795916375578145e-8 \tag{17}$$

5 `Approx(pi,10^-9)`
Evaluation time : 30.25

$$\text{Stopped by user interruption.} \tag{18}$$

6 `Approx(e,10^-3)`

$$[2,2,5,55], \frac{299}{110}, 0.10001027722705835e-3 \tag{19}$$

7 `Approx(e,10^-6)`
Evaluation time : 2.68

$$[2,2,5,55,9999],\frac{271801}{99990},0.27622704124041775e-9 \tag{20}$$

8 `Approx(e,10^-10)`
Evaluation time : 37.77

Stopped by user interruption. (21)

On peut lire par exemple que

$$\pi \approx 3+\frac{1}{8}+\frac{1}{61}+\frac{1}{5020}=\frac{1924037}{612440} \quad \text{par défaut à } 10^{-6} \text{ près}$$

l'erreur réelle étant en fait $< 10^{-8}$. On remarquera que les calculs d'une approximation par défaut de π à 10^{-9} et de e à 10^{-10}, trop longs, ont été abandonnés. Par contre, ces calculs sont rapides avec « Xcas » en utilisant l'algorithme rapide :

1

```
Approx(x,E):={
local a,d,L,m;
a:=floor(x);
d:=x-a;
L:=[a];
while(d>=E)
{m:=-floor(-1/d);
L:=append(L,m);
d:=d-1/m;
a:=a+1/m;
};
d:=evalf(d);
return L,a,d;
}
:;
```

// Parsing Approx
// Success compiling Approx

Done (22)

2 `Approx(pi,10^-9) ;`

$$[3,8,61,5020,128541454],\frac{123659257071119}{39361964043880},0.44348723577672205e-16 \tag{23}$$

3 `Approx(pi,10^-20) ;`

$$[3,8,61,5020,128541454,22548563280487732],\frac{69708464582155671201052376 3997}{22188893427192774154336242 0040},0.0000000000000000 \tag{24}$$

4 `Approx(e,10^-10)`

$$[2, 2, 5, 55, 9999, 3620210482], \frac{245994207329518}{90496211523795}, 0.96091245314812420e - 20 \tag{25}$$

Le résultat (24) signifie que

$$\frac{69708464582155671201052376399 7}{22188893427192774154336242004 0}$$

est une valeur approchée si proche de π que « Xcas » ne les distingue pas.

3.3 Algorithmes dans le cas $f(x) = x^2$

Il paraît maintenant inutile de détailler les algorithmes en langage naturel.

Approximation lente

1

```
Approx(x,E):={
local a,j,d,L,c,C;
a:=floor(x);
d:=x-a;
j:=1;
L:=[a];
C:=[];
while(d>=E)
{j:=j+1;
if(a+1/j^2<=x)
{c:=0;
while(a+1/j^2<=x)
{a:=a+1/j^2;
c:=c+1;
}
d:=x-a;
L:=append(L,j);
C:=append(C,c);
};
};
d:=evalf(d);
return L,C,a,d;
}
:;
```

// Parsing Approx
// Success compiling Approx

Done (26)

2 `Approx(2/3,10^-5) ;`

$$[0,2,3,5,9,18,90],[2,1,1,1,1,1],\frac{2}{3},0.00000000000000000 \tag{27}$$

3 `Approx(pi,10^-9) ;`
Evaluation time : 2.61

$$[3,3,6,20,71,431,11111],[1,1,1,1,1,1],\frac{1307464501751743637789}{416178876742054275600},0.83200113465409231e{-}12 \tag{28}$$

4 `Approx(pi,10^-15)`
Evaluation time : 17

Stopped by user interruption. (29)

5 `Approx(e,10^-6)`

$$[2,2,3,4,5,15,67,535],[2,1,1,1,1,1,1],\frac{502937642201}{185020419600},0.12385377612389448e-7 \tag{30}$$

6 `Approx(e,10^-9)`
Evaluation time : 2.71

$$[2,2,3,4,5,15,67,535,8986],[2,1,1,1,1,1,1,1],\frac{10152826873311159749}{3735016276465760400},0.12003731342247193e{-}11 \tag{31}$$

7 `Approx(e,10^-12)`
Evaluation time : 23.06

Stopped by user interruption. (32)

Par exemple, (27) signifie que

$$\frac{2}{3}=2\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{5^2}+\frac{1}{9^2}+\frac{1}{18^2}+\frac{1}{90^2}$$

(31) est le deuxième exemple donné au début de ce papier.

Approximation rapide

1

```
Approx(x,E):={
local a,L,d,dvalf,m,c,C;
a:=floor(x);
L:=[a];
C:=[];
d:=x-a;
while(d>=E)
```

```
{m:=-floor(-1/sqrt(d));
m:=evalf(m);
m:=floor(m);
L:=append(L,m);
c:=floor(m^2*d);
c:=evalf(c);
c:=floor(c);
C:=append(C,c);
d:=d-c/m^2;
a:=a+c/m^2;
};
dvalf:=evalf(d);
return L,a,dvalf;
}
:;
```

// Parsing Approx
// Success compiling Approx

Done (33)

2 `Approx(pi,10^-15)`

$$[3,3,6,20,71,431,11111,1096452],\frac{5457788058908923020638016630887}{173726789584655104906660552980 0},0.58273603192157406e{-}19 \quad (34)$$

3 `Approx(e,10^-12)`

$$[2,2,3,4,5,15,67,535,8986,912821],\frac{8459763577194721739044916632109}{311217309722091600545724737640 0},0.26212366721317031e{-}17 \quad (35)$$

3.4 Algorithmes dans le cas $f(x) = \sin(\pi x/2)$

Nous nous limitons ci-dessous à l'algorithme rapide :

1

```
Approx(x,E):={
local a,L,d,dvalf,m,c,C;
a:=floor(x);
L:=[a];
C:=[];
d:=x-a;
while(d>=E)
{m:=-floor(-pi/(2*asin(d)));
m:=evalf(m);
m:=floor(m);
L:=append(L,m);
```

```
c:=floor(d/sin(pi/(2*m)));
c:=evalf(c);
c:=floor(c);
C:=append(C,c);
d:=d-c*sin(pi/(2*m));
a:=a+c*sin(pi/(2*m));
a:=evalf(a);
};
dvalf:=evalf(d);
return L,C,a,dvalf;
}
:;
```

// Parsing Approx
// Success compiling Approx

$$\text{Done} \tag{36}$$

2 `Approx(0.5,10^-1)`

$$[0,3],[1],\frac{1}{2},0.00000000000000000 \tag{37}$$

3 `Approx(e,10^-9)`

$$[2,2,141,45037],[1,1,1],2.7182818281400549,0.31899025267615637e-9 \tag{38}$$

4 `Approx(pi,10^-15)`

$$[3,12,142,331322,141834644451],[1,1,1,1],3.1415926535897936,0.47731714077290871e-22 \tag{39}$$

(39) est le troisième exemple donné au début de ce papier.

Références

[1] R. Moché. *Réels et sommes de termes en 1/n*
http://gradus-ad-mathematicam.fr/TSSpecialiteMaths3.htm