AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page1

L'algorithmique n'est pas la programmation !

Roger Cuppens (IRES de Toulouse)

Résumé.

Cet article utilise la notion de fonction récursive pour introduire l'algorithmique en mathématique. Il étudie deux exemples simples pour montrer que l'algorithmique ne suffit pas pour comprendre toutes les difficultés pour effectuer des calculs mathématiques avec des calculatrices ou des ordinateurs et de comprendre les résultats ainsi obtenus.

Mots clés : *algorithmique, programmation, fonction récursive, factorielle, série harmonique***.**

1. Introduction

On sait que les nouveaux programmes de mathématiques contiennent une introduction à l'algorithmique. Si on admet la nécessité de cette introduction comme étape préliminaire à la compréhension et à l'utilisation des ordinateurs, on peut se poser la question de savoir si les techniques employées actuellement sont les plus simples et les mieux adaptées pour ce but. Dans cet article on présente un modèle montrant que l'on peut en douter.

2. Le calculable

Dans une ancienne version du Petit Larousse illustré, j'ai trouvé ces définitions d'un algorithme et de l'algorithmique :
– **algorithme** : suite finie d'opérations élémentaires constituant un schéma de calcul ou de résolution de problème ;
– **algorithmique** : science des algorithmes utilisés notamment en informatique.
Mais qu'est-ce-qu'un calcul ? qu'est-ce-qu'un problème ?

Une étude historique montre que jusqu'à la fin du 19e siècle la notion d'algorithme semble se confondre avec la notion de calcul mathématique. Mais la définition de cette dernière reste une énigme (sans gros intérêt à l'époque).

Le développement du calcul différentiel et intégral a montré la nécessité d'une définition précise de l'ensemble des réels, voire une nouvelle conception des mathématiques. Pour satisfaire ce besoin, Cantor introduit un système qui est qualifié par Hilbert comme « le paradis ». Néanmoins ce système comprend un axiome appelé *axiome du choix* qui permet de démontrer des résultats contraires au bon sens : par exemple Banach montre comment on peut découper une boule en un nombre fini de morceaux à partir desquels on peut reconstruire une boule deux fois plus grosse.

Ceci ne peut satisfaire de nombreux mathématiciens, en particulier les mathématiciens « appliqués » qui considèrent que leurs travaux doivent pouvoir

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page2

être utilisés dans d'autres sciences. Pour essayer de concilier tout le monde, Henri Poincaré a proposé de définir le *calculable*. Dans les années 1930, plusieurs solutions ont été proposées qui s'avèrent toutes équivalentes. Nous n'en retiendrons que deux :

– la machine (théorique) de Turing qui est la plus connue ;

– le λ-calcul de Church : un calcul est une fonction (ou une boite noire) où on entre des données et qui fournit un (le ?) résultat. Cette fonction peut être récursive (définie à partir d'elle). Bien entendu il faut introduire des conditions pour que le système soit cohérent.

On sait que la machine de Turing a été la base des calculateurs (pour les anglo-saxons), mais que nous appelons (malencontreusement peut être !) ordinateurs.

Par contre le point de vue de Church a inspiré les langages fonctionnels dont le premier fut LISP, mais aussi le langage LOGO développé pour apprendre les idées de base de l'informatique à de très jeunes enfants. Ces langages ont été très décriés car ils nécessitent des machines ayant beaucoup de mémoire, ce qui n'est évidemment plus un problème actuellement. En s'inspirant de ce modèle, on peut introduire la définition :

Un *algorithme* est une fonction qui à partir de *variables* (les données du problème) fournit une *réponse* (la solution du problème).

La fonction peut faire appel à d'autres fonctions (sous-problèmes) et éventuellement à elle-même mais avec d'autres variables (appel récursif).

Remarques. 1. Ceci concerne la plupart des problèmes que l'on se pose : calculer le prix d'un achat, construire un itinéraire, construire une figure géométrique, etc.
2. Tous ces problèmes peuvent se faire actuellement soit « à la main » pour les plus simples, soit avec une machine. Dans ce dernier cas, on distinguera soigneusement
– l'algorithme (description générale de la méthode)
– les développements nécessaires pour utiliser cette méthode sur la machine (programmation)

Pour écrire un algorithme, l'emploi de fonctions récursives écrites avec la syntaxe habituelle semble le mieux adapté et le plus conforme à l'usage des mathématiciens.

3. Une algorithmique de base

Il est traditionnel de partager les connaissances en divers *mondes* (ou *univers*) définis par quelques notions primitives de base, constantes ou fonctions.

3.1. Le monde de la logique.

Dans le monde du calcul (et ailleurs), on ne peut rien faire sans un minimum de logique. Dans le monde de la logique, on admet deux constantes de base, *vrai* et *faux,* et on appelle *prédicats* (ou *fonctions booléennes*) des fonctions

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page3

ayant pour valeurs ces deux constantes. On utilise une primitive de base, à savoir la fonction *si ... alors ... sinon* dont la syntaxe est :

si (*prédicat*) alors [fonction 1] sinon [*fonction 2*].

Avec cette seule primitive, on peut définir les principales fonctions logiques : *non*, *et*, *ou, ou_exclusif* :

non (p) = si (p) alors [faux] sinon [vrai]
et (p,q) = si (p) alors [q] sinon [faux]
ou (p,q) = si (p) alors [vrai] sinon [q]
ou_exclusif (p,q) = si (p) alors [non(q)] sinon [q]

3.2. Le monde des listes

Dans cette algorithmique, la structure fondamentale est celle de *liste* qui correspond dans le langage mathématique traditionnel à celle de suite finie. Une liste l sera notée comme une suite d'éléments dans des crochets

$l = [a\ b\ c\ ...]$

Pour les manipuler, on introduit
– une liste vide notée [ ] ;
– un prédicat de base *videp?* : pour une liste l,

videp? (l) = si (l = [ ]) alors [vrai] sinon [faux] ;

– un constructeur *metspremier* : si x est un élément et l une liste, metspremier(x,l) fournit la liste composée de l'élément x suivi de tous les éléments de la liste l.
– deux sélecteurs *premier* et *saufpremier* ; comme leur nom l'indique, pour une liste l non vide, premier (l) fournit le premier élément de l et saufpremier (l) fournit la liste l amputée de son premier élément.

Il est commode aussi d'introduire des fonctions *dernier* et *saufdernier*, mais on constatera qu'on peut les redéfinir à partir des précédentes : pour une liste non vide l,

dernier (l) = si videp? (saufpremier (l))
alors [premier l)]
sinon [dernier (saufpremier (l))]

saufdernier (l) = si videp? (saufpremier (l))
alors [l]]
sinon [metspremier (premier (l), saufdernier (l))]

3.3. Les mondes numériques.

Dans ces mondes, on suppose un ensemble de nombres sur lesquels on sait définir (au moins) :
– les nombres 0 et 1,
– les opérations de base + et ×,
– les relations d'ordre <, >, ≤ et ≥.

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page4

Remarque. On utilisera les notations traditionnelles, mais pour les puristes on pourrait introduire la notation polonaise : + (a,b) au lieu de a + b etc.

Voici quelques exemples de fonctions utiles :

–si l est une liste de nombres, les fonctions

sommeliste (l) = si videp? (l) alors [0]
sinon [premier (l) + sommeliste (saufpremier (l))]
produitliste (l) = si videp? l) alors [1]
sinon [premier (l) × produitliste (saufpremier (l)]

fournissent les opérations notées traditionnellement ∑ et ∏.

– si p et q sont deux entiers, la fonction

listenombres (p,q) = si ($q < p$)
alors [[ ]]
sinon [metspremier (p,listenombres (p+1,q)]

fournit la liste croissante des nombres de p à q.

4. Un exemple : le calcul de la factorielle

4.1. Généralités

En arithmétique, on définit la factorielle d'un nombre positif n (notée traditionnellement n!) comme étant le produit de la liste des n premiers nombres entiers, autrement dit :

$$n! = \prod_{j=1}^{n} j$$

Puisque pour n = 0, la liste des entiers est vide, on peut admettre que

$$0! = 1. \tag{1}$$

De plus l'associativité et la commutativité de la multiplication donnent

$$n! = \prod_{j=1}^{n} j = \prod_{j=1}^{n-1} j \times n = n \times \prod_{j=1}^{n-1} j = n \times (n-1)! \tag{2}$$

En algorithmique, on préférera introduire la notation fonctionnelle factorielle (n) en lieu et place de n!. La définition donnée ci-dessus nous permet de dire que

$$\boxed{\text{factorielle}(n) = \text{produitliste}(\text{listenombres}(1,n))}$$

Néanmoins on préférera prendre comme définition

$$\boxed{\begin{array}{lcll} \text{factorielle}(n) & = & \text{si } (n=0) & \text{alors} \quad [1] \\ & & & \text{sinon} \quad [n \times \text{factorielle}(n-1)] \end{array}} \tag{3}$$

qui se déduit immédiatement de (1) et (2). On obtient alors facilement « à la

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page5

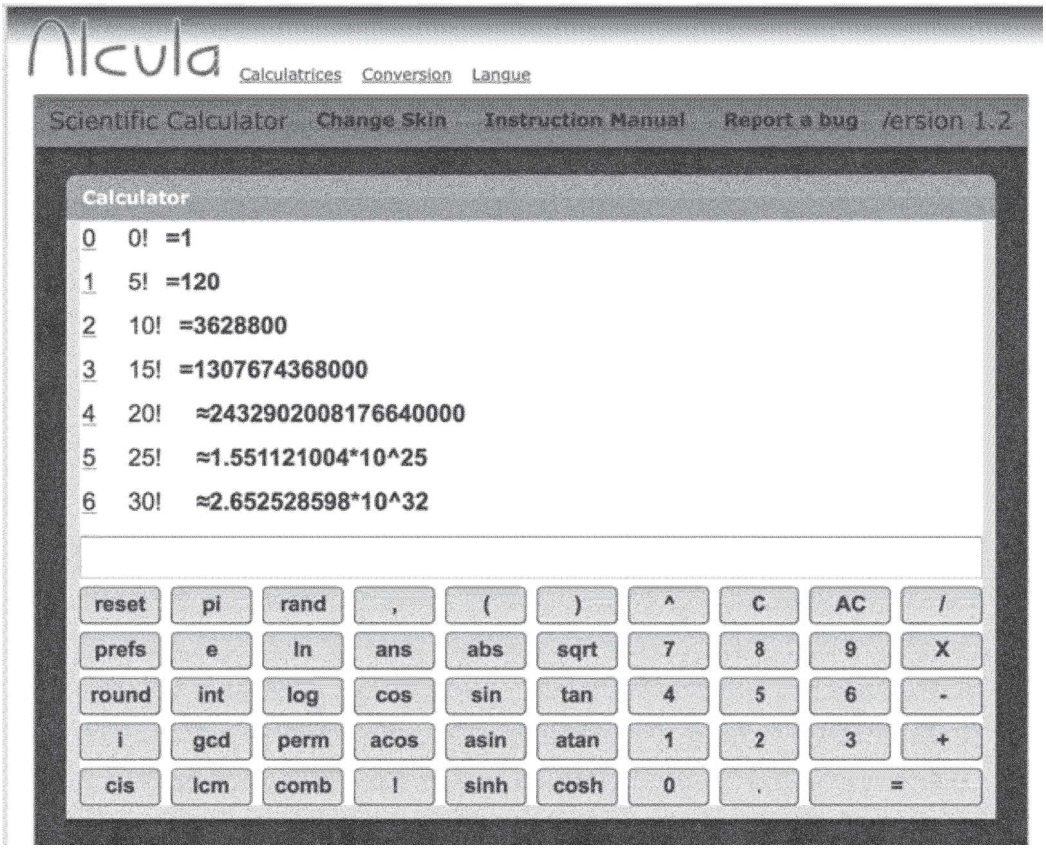
main » la valeur de factorielle (n) pour les petites valeurs de n :

n	0	1	2	3	4	5	6	7	8
factorielle(n)	1	1	2	6	24	120	720	5040	40320

Mais ceci devient rapidement fastidieux. Pour de plus grandes valeurs, on peut utiliser une calculatrice ou un ordinateur.

4.2. Avec une calculatrice

Il existe des calculatrices scientifiques avec une touche « ! » donnant immédiatement la « valeur » d'une factorielle, par exemple la calculatrice Alcula que j'ai trouvée sur mon ordinateur et avec laquelle en tapant le nombre n et en cliquant sur les touches « ! » et « = » on obtient immédiatement une réponse :



Les premières réponses ;

1! =1	7! =5040	13! =6227020800
2! =2	8! =40320	14! =87178291200
3! =6	9! =362880	15! =1307674368000
4! =24	10! =3628800	16! =20922789888000
5! =120	11! =39916800	17! =355687428096000
6! =720	12! =479001600	18! =6402373705728000

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page6

sont des nombres entiers jusque 18 (qui confirment évidemment les valeurs obtenues précédemment à la main). Puis de 19 à 21 la calculatrice semble hésiter puisque sa réponse est un nombre entier mais précédé du symbole « ≈ » au lieu du symbole « = » :

19! ≈121645100408832020

20! ≈2432902008176640000

21! ≈51090942171709440000

On remarquera que la valeur fournie pour 19 ne peut pas être exacte car elle doit se terminer comme pour 18 par 000. Ensuite de 22 à 170 elle fournit une valeur approchée avec la notation habituelle des nombres réels $p \times 10^e$ qui fournit une information sur le nombre x cherché : x est un nombre entier de $e + 1$ chiffres dont les 10 premiers sont les chiffres composant le nombre p :

22! ≈1.124000728*10^21

23! ≈2.585201674*10^22

24! ≈6.204484017*10^23

25! ≈1.551121004*10^25

30! ≈2.652528598*10^32

40! ≈8.159152832*10^47

50! ≈3.04140932*10^64

60! ≈8.320987113*10^81

110! ≈1.588245542*10^178

120! ≈6.689502913*10^198

130! ≈6.466855489*10^219

140! ≈1.346201248*10^241

26! ≈4.032914611*10^26

27! ≈1.088886945*10^28

28! ≈3.048883446*10^29

29! ≈8.841761994*10^30

70! ≈1.197857167*10^100

80! ≈7.156945705*10^118

90! ≈1.485715964*10^138

100! ≈9.332621544*10^157

150! ≈5.713383956*10^262

160! ≈4.714723636*10^284

170! ≈7.257415615*10^306

171! ≈Infinity

Enfin à partir de 171, elle abandonne.

En résumé la calculatrice traite différemment les petits et les grands nombres entiers. Pour la factorielle elle fournit jusque 170 une valeur qui suivant les cas est le nombre entier exact ou un ordre de grandeur de ce nombre.

4.3. Avec un ordinateur

Pour exécuter un calcul avec un ordinateur, il faut commencer par choisir le logiciel approprié, puis apprendre l'algorithme correspondant dans le langage de la machine. C'est cette dernière étape que l'on appelle la *programmation*. Dans le cas qui nous occupe, le langage doit connaître l'arithmétique et pouvoir traiter des grands nombres entiers. Pour cela j'utilise le langage x-logo. Dans ce langage, voici le programme ad hoc :

```
pour fact :n
si :n=1 [ret 1] [ret :n * fact :n-1]
fin
```

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page7

(la primitive *ret* indique à la machine la valeur à retourner).

Avec cette procédure, on obtient bien entendu les 18 premières valeurs de fact (n), puis

$$\text{fact } (19) = 121645100408832000$$
$$\text{fact } (20) = 2432902008176640000$$
$$\text{fact } (21) = 51090942171709440000$$

La première corrige l'erreur signalée dans les résultats de la calculatrice tandis que les deux autres sont semblables à ceux de cette dernière. Si l'on continue, on obtient

$$\text{fact } (22) = 1124000727777608000000 \quad (4)$$
$$\text{fact } (23) = 25852016738884980000000 \quad (5)$$

Ces valeurs sont manifestement incorrectes car comme fact (21) elles devraient se terminer par 4 zéros. Mais les deux sont composées d'un nombre de 16 chiffres suivis de 6 zéros pour le premier et de 7 zéros pour le deuxième. Or 16 + 6 et 16 + 7 sont exactement le nombre de chiffres annoncés par la calculatrice. On peut donc penser qu'une lecture correcte de (4) et (5) est celle de la calculatrice :

$$\text{fact } (22) \approx 1{,}124000727777608 \times 10^{21}$$
$$\text{fact } (23) \approx 2585201673888498 \times 10^{22}$$

Or dans la notice de x-Logo on trouve la primitive suivante :

fixedecimales *n*

Permet de fixer le nombre de décimales souhaités lors des calculs. Cette primitive règle en fait le degré de précision des calculs. Quelques précisions :

- Par défaut, les calculs se font avec 16 décimales.
- Si n est négatif, le mode d'affichage par défaut est choisi.
- Si n est nul, les nombres affichés sont arrondis à l'unité.

Le nombre 16 est donc la valeur par défaut. On peut donc espérer les valeurs suivantes en utilisant la procédure suivante

```
pour efact :n :p
fixedecimales :p
ret fact :n
fin
```

qui donne (avec les valeurs de p fournies par la calculatrice) :

$$\text{efact } (22,21) = 112400072777760768000 0$$
$$\text{efact } (23,22) = 2585201673888497664000 0$$
$$\text{efact } (24,23) = 6204484017332394393600 00$$
$$\text{efact } (25,25) = 1551121004333098598400 0000$$

et aussi

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page8

efact (170,306) =
725741561530799896739672821112926311471699168129645137654357779
890056184340170615785235074924261745951149099123783852077666 60
225654427530253289007732075109024004302800582956039666125996 58
257104398558294257568966313439612262571094946806711205568880 45
719334021266145280000000000000000000000000000000000000000

Pour obtenir fact (n) pour des valeurs de n plus grandes, on peut déterminer le taux de précision p correspondant. En effet si x est un nombre réel de la forme $k \times 10^p$ avec $k < 10$, alors log (x) = p + log (k) et, puisque log (k) < 1, p est la partie entière de log (x). De plus de

$$\log(n!) = \sum_{j=1}^{n} \log(j)$$

on déduit qu'en fournissant à x-Logo le programme

```
pour pfact :n
ret tronque logfact :n
fin
pour logfact :n
si :n=1 [ret 0] [ret (log10 :n) + logfact :n-1]
fin
```

on obtiendra par exemple

n	1	2	3	4	5	6	7	8	9	10	11	12
pfact(n)	0	0	0	1	2	2	3	4	5	6	7	8

n	13	14	15	16	17	18	19	20	21	22	23	24
pfact(n)	9	10	12	13	14	15	17	18	19	21	22	23

mais aussi

n	180	190	200	250	300	350	400
pfact(n)	329	351	374	492	614	740	868

puis

efact (200,374) =
788657867364790503552363213932185062295135977687173263294742 53
324435944996340334292030428401198462390417721213891963883025 76
427902426371050619266249528299311134628572707633172373969889 43
922445621451664240254033291864131227428294853277524242407573 90
324032125740557956866022603190417032406235170085879617892222 27
896237038973747200000000000000000000000000000000000000000000
000.

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page9

Remarque. On peut calculer une valeur exacte de la factorielle de n sans avoir la valeur exacte de pfact (n). En effet on peut utiliser le fait que pour deux entiers différents k et l $(k < l)$,

– si efact(n,k) < efact(n,l), alors $k <$ pfact (n) ;

– si efact(n,k) = efact(n,l), alors $k \geq$ pfact (n) et efact(n,k) = efact$(n,$pfact $(n))$.

Par exemple de

efact (200,300) =

78865786736479050355236321393218506229513597768717326329474253
32443594499634033429203042840119846239041772121389196388302576
42790242637105061926624952829931113462857270763317237396988943
92244562145166424025403329186413122742829485327752424240757390
32403212574055795686602260319041703240623517008588000000000000
00000000000000000000000000000000000000000000000000000000000000
000

efact (200,400) = efact (200,500) =

78865786736479050355236321393218506229513597768717326329474253
32443594499634033429203042840119846239041772121389196388302576
42790242637105061926624952829931113462857270763317237396988943
92244562145166424025403329186413122742829485327752424240757390
32403212574055795686602260319041703240623517008587961789222227
89623703897374720000000000000000000000000000000000000000000000
000

on déduit que 300 < pfact (200) < 400 et que 200! = efact (200,400).

4.4. Remarques

1. Le calcul d'un nombre entier peut fournir deux réponses : le nombre exact ou l'ordre de grandeur de ce dernier.

2. La réponse d'une machine peut être difficile à interpréter, voire être incorrecte. Dans ce dernier cas, on invoquera que l'algorithme est faux ou que l'on est sorti de son domaine de validité.

3. L'algorithmique des entiers naturels donne une idée très différente de celle de l'ensemble $\mathbb{N}$ habituel. Par exemple, cet ensemble doit être considéré comme un ensemble infini potentiel comme chez Euclide. De plus un théorème comme « tout entier est un produit de nombres premiers » ne peut fournir qu'un algorithme de domaine limité, ce qui est d'ailleurs utilisé en théorie du codage.

5. La suite harmonique

5.1. Introduction

Mon point de départ est la deuxième partie de l'épreuve du Bac S " Algorithmique " Métropole 2012 :

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page10

Partie B

Soit (u_n) la suite définie pour tout entier strictement positif par

$$u_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} - \ln n.$$

1. On considère l'algorithme suivant :

Variables :	i et n sont des entiers naturels. u est un réel.
Entrée :	Demander à l'utilisateur la valeur de n.
Initialisation :	Affecter à u la valeur 0.
Traitement :	Pour i variant de 1 à n. Affecter à u la valeur $u + \frac{1}{i}$
Sortie :	Afficher u.

Donner la valeur exacte affichée par cet algorithme lorsque l'utilisateur entre la valeur $n = 3$.

2. Recopier et compléter l'algorithme précédent afin qu'il affiche la valeur de u_n lorsque l'utilisateur entre la valeur de n.

3. Voici les résultats fournis par l'algorithme modifié, arrondis à 10^{-3}.

n	4	5	6	7	8	9	10	100	1 000	1 500	2 000
u_n	0,697	0,674	0,658	0,647	0,638	0,632	0,626	0,582	0,578	0,578	0,577

À l'aide de ce tableau, formuler des conjectures sur le sens de variation de la suite (u_n) et son éventuelle convergence.

Plutôt que de critiquer *in abstracto* cet extrait, je vais montrer d'abord ce que peuvent apporter mes idées et les difficultés qu'on peut rencontrer.

5.2. Définition et étude dans l'ensemble des rationnels

Pour un professeur de mathématique, il s'agit ici de l'étude de la notion de série harmonique présentée par exemple dans la rubrique de ce nom de Wikipédia comme la somme des inverses des premiers nombres entiers, autrement dit la suite des nombres

$$h_n = \sum_{j=1}^{n} \frac{1}{j}$$

ou si on pense algorithmiquement de la fonction récursive

$$\boxed{\text{harmo}(n) = \text{si } (n = 1) \text{ alors } [1] \text{ sinon } \left[\text{harmo}(n-1) + \frac{1}{n}\right]} \qquad (6)$$

Cette fonction est particulièrement importante car c'est la plus simple des sommes dont les termes tendent vers 0, mais qui tend vers l'infini (autrement dit qui devient arbitrairement grande).

Peut-on « voir » ceci en calculant h_n ? On a vu que cette question peut avoir plusieurs réponses suivant que l'on veut une valeur « exacte » de ce nombre ou seulement un ordre de grandeur. Pour une valeur exacte, on peut considérer que la fonction harmonique prend ses valeurs dans l'ensemble des rationnels et, puisque cet ensemble est un corps, on a le théorème suivant rarement énoncé, mais facile à démontrer :

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page11

Si (a_j) et (b_j) sont deux suites de nombres entiers avec $b_j \neq 0$, alors $\sum_{j=1}^{n} \frac{a_j}{b_j} = \frac{p_n}{q_n}$ avec $q_n = \prod_{j=1}^{n} b_j$ et $p_n = b_n\, p_{n\text{-}1} + a_n\, q_{n\text{-}1}$.

En appliquant ce résultat pour $a_j = 1$ et $b_j = j$, on déduit que $\mathrm{harmo}\,(n)$ est un nombre rationnel de dénominateur $n!$ et de numérateur le nombre $\mathrm{numharmo}\,(n)$ défini par

$$\boxed{\mathrm{numharmo}(n) = \text{ si } (n=1) \text{ alors } [1] \text{ sinon } [n \times \mathrm{numharmo}(n-1) + \mathrm{fact}(n-1)]}$$

On obtient facilement les premières valeurs de cette fonction :

n	1	2	3	4	5	6	7	8
$\mathrm{numharmo}(n)$	1	3	11	50	274	1764	13068	109584

Remarque. Cette méthode est utile lorsque l'on s'intéresse aux problèmes de la précision (ou des erreurs) de calculs faits par une machine. En effet dans un tel calcul sur les entiers on peut faire des additions, soustractions et multiplications exactes ; seules les divisions introduisent de telles erreurs qui se multiplieraient si on faisait le calcul de la fonction harmo à partir de (6) alors que la méthode proposée ici ne comprend qu'une seule division terminale. Pour de tels exemples, voir dans [2] le calcul des décimales des constantes e et π. Ici pour étudier les propriétés de la suite harmo, on n'a pas besoin d'une grande précision.

5.3. Calcul et propriétés de la suite harmonique

Avec les méthodes du paragraphe 4, on obtient de la définition (6) les valeurs approximées à 10^{-3} suivantes :

n	1	2	3	4	5	6	7	8	9	10
$\mathrm{harmo}(n)$	1	1.5	1.833	2.083	2.283	2.45	2.593	2.718	2.829	2.929

n	20	30	40	50	60	70	80	90	100
$\mathrm{harmo}(n)$	3.598	3.995	4.279	4.499	4.68	4.832	4.965	5.083	5.187

n	200	300	400	500	600	700	800	900	1000
$\mathrm{harmo}(n)$	5.878	6.283	6.57	6.793	6.975	7.129	7.262	7.38	7.485

On voit que la fonction croissante $\mathrm{harmo}(n)$ a une croissance extrêmement lente puisque $\mathrm{harmo}\ (1000) - \mathrm{harmo}\ (1) \simeq 6.485$. Or on connait une fonction ayant un comportement semblable, à savoir la fonction $\ln$ (logarithme népérien) pour laquelle $\ln\ (1000) \simeq 6.908$.

On va donc comparer ces deux fonctions en introduisant une fonction que pour des raisons qui apparaîtront ci-dessous nous appelons gamma :

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page12

$$\boxed{\text{gamma}(n) = \text{harmo}(n) - \ln(n)}$$

avec laquelle on obtient les valeurs :

n	1	2	3	4	5	6	7	8
$\text{gamma}(n)$	1	0.807	0.735	0.697	0.674	0.658	0.647	0.638

(approchées à 10^{-3}) et pour les dernières valeurs

n	1000	1500	2000	2500	3000	3500
$\text{gamma}(n)$	0.577716	0.577549	0.577466	0.577416	0.577382	0.577359

que ma machine a bien voulu me fournir (approchées à 10^{-6}). On peut en conjecturer que la suite $\text{gamma}(n)$ est décroissante et qu'elle a donc à l'infini une limite γ qui s'appelle dans la littérature constante d'Euler et dont 0.577 serait une valeur approchée à 10^{-3}.

5.4. La série harmonique alternée

On peut traiter de même la série harmonique alternée définie par la suite

$$k_n = \sum_{j=1}^{n} \frac{(-1)^j}{j}$$

ou la fonction

$$\boxed{\text{harmoalt}(n) = \text{si } (n=1) \text{ alors [-1] sinon } \left[\text{harmoalt}(n-1) + \frac{(-1)^n}{n}\right]}$$

Puisque les suites (k_{2n}) et (k_{2n+1}) sont adjacentes, la suite (k_n) converge et de

$$\text{harmoalt}\ (3000) \simeq -0.692581,\ \text{harmoalt}\ (3001) \simeq -0.693314$$

et $\exp(0.693) \simeq 2$, on peut conjecturer que la limite de (k_n) est ln 2, ce qui peut se démontrer à partir de de la convergence de la fonction gamma.

5.5.

Je terminerai ce paragraphe par quelques remarques sur le sujet du Baccalauréat cité au début de ce paragraphe :

Dès la première question, sans parler du regrettable emploi de la variable u alors que u_n vient juste d'être employé pour une toute autre chose, on constate toute l'ambiguïté de la notion d'algorithme adoptée (plus proche à vrai dire d'une programmation) puisqu'on dit « u est un réel » et on ne précise pas la notion de « valeur exacte » pour un tel nombre : 11/6 ou 1,833… ?

La deuxième question montre que la notion d'algorithme utilisée ne convient pas pour élaborer un calcul complexe puisqu'elle suggère de recopier au lieu d'utiliser un calcul déjà fait.

AlgorithmeIRES1.qxp_Mise en page 1 22/11/2017 09:52 Page13

6. Conclusion

Avant l'avènement des ordinateurs, les calculs se faisaient « à la main » en utilisant des tables de logarithmes et trigonométriques et /ou des règles à calcul. Dans les années soixante, la quatrième épreuve de l'agrégation de mathématiques était une épreuve de calcul numérique de quatre heures se faisant dans ces conditions et dont le but était un calcul complexe avec estimation de la précision du résultat obtenu. On aurait pu croire que l'utilisation des portables et des calculatrices scientifiques changerait beaucoup les choses. Il n'en a rien été : les discussions sur ces moyens de calcul portaient plus sur la possibilité de fraudes aux examens et la préservation de l'égalité des chances aux examens que leur utilisation dans l'enseignement. Peut-on croire que l'introduction actuelle de « l'algorithmique » sans parler des nombreux problèmes posés par les calculs sur machine dont je viens de montrer quelques exemples dans cet article changera réellement les chose ?

Références

[1] Roger CUPPENS. La récursivité ou l'algorithmique sans boucles. Bulletin de l'APMEP nº 513 (2015, p. 211-226.

[2] Roger CUPPENS. La récursivité de la tortue. Bulletin de l'APMEP nº 515 (2015) p. 455-464.

[3] Roger CUPPENS. Les reptiles et le pavage du plan. Atelier D31 présenté aux Journées de l'APMEP 2015 de Laon.

[4] Roger CUPPENS. La tortue, les reptiles et les dragons : le pavage du plan par des fractales. Atelier L32 présenté aux Journées de l'APMEP 2015 de Laon.

[5] Roger CUPPENS. Les reptiles d'ordre 2. Atelier présenté aux Journées de l'APMEP 2016 de Lyon.

Ces trois derniers textes restés inédits peuvent être obtenus sur simple demande auprès de l'auteur : roger.cuppens @ orange.fr

Je remercie Michel Carral et Tony Paintoux qui ont bien voulu relire ce texte et me faire des suggestions importantes.