Outils pour explorer les mathématiques numériques

François Parisot

Je présente quelques algorithmes qui permettent de trouver par expérimentations quelques propriétés de la représentation des nombres utilisés par une calculatrice ou un ordinateur et des conséquences du caractère fini de cette représentation.

Je pense qu'ils pourraient être utilisés l'an prochain avec des élèves de première pour prolonger le travail de seconde sur l'introduction de l'algorithmique et améliorer leur connaissance des outils de calcul numérique.

Les algorithmes ont été testés sur des calculatrices Casio Graph 25 et Graph 95, TI 82-stats et sur mon PC avec le tableur d'OpenOffice et en écrivant des commandes JavaScript dans la barre d'adresse du navigateur.

Propriétés de la représentation des nombres

On suppose que la machine stocke pour chaque nombre x un signe s, une mantisse (significande) m et un exposant e, tels que $x = \text{s} \times m \times b^e$ où $s \in \{+1 ; -1\}$, $m \in [1 ; b[$, $e \in \mathbb{Z}$ et b est la base de numération.

Les microprocesseurs utilisent des registres de taille finie exprimée en octets (huit chiffres binaires ou bits). Le nombre de chiffres de la mantisse est donc nécessairement fini : soit on travaille en base 2 et l'unité d'occupation mémoire est le bit, soit on travaille en base 10 (système décimal codé binaire ou BCD) et chaque chiffre du système décimal est codé sur un demi-octet.

La taille finie des mantisses a pour conséquence immédiate que si le nombre x est très grand par rapport au nombre y, la somme $x + y$ aura la même représentation en mémoire que x.

Recherche de la base utilisée par la machine

D'après http://images.math. cnrs.fr/Erreurs-en-arithmetique-des.html

Variables
  a, b, de type numérique
Initialisation
  a prend la valeur 1
  b prend la valeur 1
Traitement
  tant que $((a + 1) - a) - 1 = 0$ faire
    a prend la valeur $2a$
  tant que $((a + b) - a) - b \neq 0$ faire
    b prend la valeur $b + 1$
Sortie
  Affiche b

Algorithme 1 : recherche de la base

Un exemple pour comprendre le fonctionnement

La première boucle cherche le premier nombre 2^n tel que l'addition de 1 à la représentation en mémoire ne modifie pas la valeur stockée.

Pour une machine calculant en base 10 avec trois chiffres pour les mantisses, le tableau suivant montre le contenu des mémoires après chaque opération :

a	mantisse	exposant	$a+1$	$(a+1)-a$	$((a+1)-a)-1$
1	1	0	2	1	0
2	2	0	3	1	0
4	4	0	5	1	0
8	8	0	9	1	0
16	1.6	1	17	1	0
32	3.2	1	33	1	0
64	6.4	1	65	1	0
128	1.28	2	129	1	0
256	2.56	2	257	1	0
512	5.12	2	513	1	0
1 020	1.02	3	1 020	0	–1

On remarque que 1 024 et 1 021 ont la même représentation 1.02×10^3 qui vaut 1 020.

Suivons maintenant la deuxième boucle qui cherche le plus petit entier à ajouter au contenu de la variable a pour modifier sa valeur d'autant que lui :

si la mantisse est tronquée

b	$a+b$	$(a+b)-a$	$((a+b)-a)-b$
1	1 020	0	–1
2	1 020	0	–2
3	1 020	0	–3
4	1 020	0	–4
5	1 020	0	–5
6	1 020	0	–6
7	1 020	0	–7
8	1 020	0	–8
9	1 020	0	–9
10	1 030	10	0

si la mantisse est arrondie

b	$a+b$	$(a+b)-a$	$((a+b)-a)-b$
1	1 020	0	–1
2	1 020	0	–2
3	1 020	0	–3
4	1 020	0	–4
5	1 030	10	5
6	1 030	10	6
7	1 030	10	7
8	1 030	10	8
9	1 030	10	9
10	1 030	10	0

La programmation de cet algorithme sur une calculatrice donne la réponse 10 alors que la traduction dans une feuille de calcul d'un tableur sur un ordinateur donne la réponse 2.

On peut également obtenir la réponse 2 avec la ligne suivante dans la zone d'adresse d'un navigateur :

javascript :a=l ; b=l ; while(((a+l)-a)-l==0) a*=2 ; while(((a+b)-a)-b !=0) b+=l ; alert(b)

On peut en déduire que le tableur et JavaScript utilisent le système binaire et les calculatrices un système décimal codé binaire.

Nombre de chiffres significatifs connus

L'algorithme suivant compte le nombre de chiffres après le séparateur « décimal » (virgule ou point) ; il donnera donc le nombre de chiffres de la mantisse si on entre un nombre d'exposant −1, c'est-à-dire de la forme $0,c_1c_2c_3\ldots$ Pour ce faire, on élimine les chiffres un par un en décalant la virgule d'un rang vers la droite puis en gardant la partie fractionnaire du nombre obtenu.

Variables
a, *x*, de type numérique
b, base de numération
n, compteur de chiffres
Entrée
Saisir *x*, entre 1/*b* et 1
Initialisation
b prend la valeur résultat de l'algorithme 1
a prend la valeur *x*
n prend la valeur 0
Traitement
tant que $a \neq 0$ faire
    a prend la valeur ab — $[ab]$ ($[x]$ désigne la partie entière de x)
    n prend la valeur $n + 1$
Sortie
Affiche *n*

Algorithme 2 : nombre de chiffres

Après avoir testé cet algorithme avec diverses entrées décimales, rationnelles ou irrationnelles, le résultat maximum est 14 pour une calculatrice TI, 15 pour une calculatrice Casio et 53 pour un ordinateur.

Traduction en Javascript :

```
a=eval (prompt ("nbre?")) ; n=0 ;while(a !=0) a=2*a-Math.floor(2*a) ; n++; alert(n) ;
```

Les calculatrices utilisent donc des mantisses de 14 ou 15 décimales et les ordinateurs utilisent des mantisses de 53 bits (information confirmée par la norme IEEE754).

Cet algorithme permet de constater déjà une particularité des calculatrices Casio. Pour une mantisse de 15 chiffres, 0,100 000 000 000 001 devrait pouvoir être stocké exactement. Or en entrant $10^{-1} + 10^{-15}$, l'algorithme programmé sur une Casio donne comme réponse 1.

Quel est le suivant de 1 ?

Pour confirmer les résultats obtenus avec l'algorithme précédent, on recherche le plus petit entier naturel n tel que $1 + b^{-n} = 1$ pour la machine.

On devrait obtenir les mêmes réponses que précédemment : 14, 15 ou 53.

Variables
b, base de numération
a, nombre « réel »
n, rang
Initialisation
b prend la valeur résultat de l'algorithme 1
a prend la valeur 1
n prend la valeur 0
Traitement
tant que $1 + a \neq 1$ faire
  a prend la valeur *a/b*
  n prend la valeur $n + 1$
Sortie
Affiche *n*

Algorithme 3 : suivant de 1

Traduction en Javascript :
n=0 ; a=l ; while(l+a !=l) a/=2 ; n+=l ; alert(n) ;

Les résultats n'étant pas ceux attendus sauf avec Javascript, on remplace successivement le test par
1. $(1 + a) - 1 \neq 0$,
2. $\ln(1 + a) \neq 0$,
3. $1 + b^{-k} + a - 1 - b^{-k} \neq 0$ où par exemple, $k = 11$ ou 12 en base 10, $k = 30$ en base 2.

Tableau des résultats obtenus :

machine	$1 + a \neq 1$	$(1 + a) - 1 \neq 0$	$\ln(1 + a) \neq 0$	$1 + b^{-k} + a - 1 - b^{-k} \neq 0$
TI	10	13	14	14
Casio 25	14	14	14	15 avec $k = 12$
Casio 95	13	13	14	15 avec $k = 11$
OOCalc	48	48	53	53 avec $k = 30$
Javascript	53	53	53	53 avec $k = 30$

Cette expérience prouve que les calculatrices et les tableurs ne comparent pas strictement les représentations internes des nombres mais sont programmés pour répondre que deux nombres sont « égaux » quand ils sont « suffisamment » proches l'un de l'autre.

Cette expérience donne une solution pour comparer avec la précision maximum de la machine des nombres que les programmeurs considèrent « trop proches ». Il suffit en effet de modifier les écarts d'ordre de grandeur en remplaçant la différence $x - y$ par $x + b^{-k} - y - b^{-k}$, b étant la base utilisée et k un entier adapté à l'ordre de grandeur de x ou y.

Conséquences de la représentation finie

La représentation finie des nombres peut générer des erreurs dans les calculs de limites ou faire donner des résultats inattendus à des algorithmes rédigés sans en tenir compte.

Arrondis et itérations

Pour vérifier si la mantisse a été arrondie ou tronquée, on regarde l'évolution d'une suite récurrente de premier terme rationnel non décimal, mathématiquement constante, mais divergente sur les machines.

Soit la suite (u_n) définie par $u_0 = \frac{2}{3}$ et $u_{n+1} = 16u_n - 10$ pour tout entier naturel n.

Calculer u_{15}.

Variables
  u, terme de la suite
  n, rang
Initialisation
  u prend la valeur 2/3
Traitement
  pour n variant de 1 à 15 faire
    u prend la valeur $16u - 10$
Sortie
  Affiche u

Algorithme 4 : suite constante !

La calculatrice TI affiche 24019.86468, la calculatrice Casio 25 affiche –4803.172936, la Casio 95 affiche 2402.586468 et sur ordinateur, le tableur ou la commande Javascript u=2/3 ; for(k=l ;k<16 ;k++) u=16*u-10 ; alert(u) ; affichent –42.

Les résultats permettent de conjecturer que la Casio Graph 25 tronque, la Graph 95 et la TI arrondissent et que l'arrondi de l'ordinateur est par défaut.

Dichotomie

On souhaite trouver les racines de la fonction f définie par

$$f(x) = x^3 - 3x^2 + 3x - 1.$$

L'algorithme de dichotomie tel qu'il est décrit habituellement devrait donner une suite d'intervalles d'amplitude tendant vers 0 et contenant la racine 1.

Variables
a, *b*, bornes
m, milieu de l'intervalle
i, indice de boucle
Entrée
Saisir *a* et *b* tels que $f(a) < 0$ et $f(b) > 0$
Traitement
pour *i* variant de 1 à 50 faire
  m prend la valeur $(a + b)/2$
  si $f(m) > 0$ alors
    b prend la valeur *m*
  sinon
    a prend la valeur *m*
Sortie
Affiche *a* et *b*

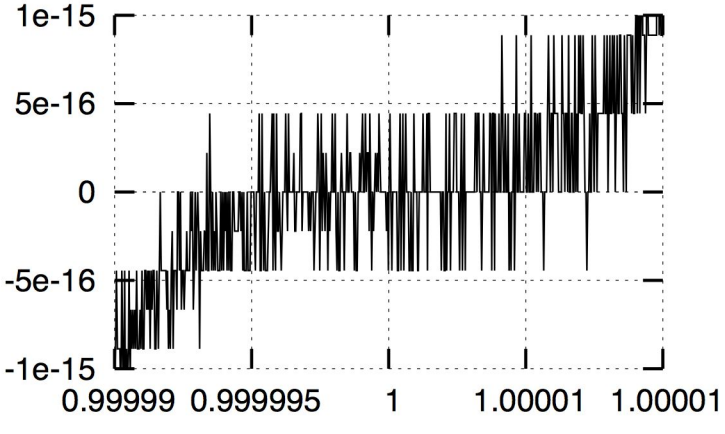
Algorithme 5 : dichotomie

Résultats inattendus

La programmation de cet algorithme montre que rapidement, *a* et *b* sont tous les deux strictement supérieurs à 1. Que calcule réellement l'algorithme ?

Pour obtenir des valeurs inférieures à 1, il suffit de remplacer le test de la condition $f(m) > 0$ par $f(m) < 0$ et d'échanger *a* et *b* dans les deux alternatives.

Les résultats ne sont pas ceux attendus car la fonction f définie par $f(x) = x^3 - 3x^2 + 3x - 1$ étudiée ici a pour une machine un grand nombre de zéros, comme le montre la représentation graphique suivante :



Je laisse au lecteur en exercice la rédaction d'algorithmes qui donneraient la plus grande ou la plus petite des « racines informatiques » de f.