LE CONTINU ET L'ORDINATEUR

Jacques HARTHONG

1.— Les systèmes de numération et les nombres.

Le système romain de numération ne permet pas d'écrire un nombre tel que un milliard. Par conséquent, l'ensemble des nombres susceptibles d'une écriture romaine est entièrement majoré par un milliard. Un théorème connu de l'arithmétique nous assure que cet ensemble possède dès lors un plus grand élément N. Quel est ce nombre N, le plus grand qui puisse s'écrire en chiffres romains?

Tout mathématicien sent qu'il y a là quelque chose qui ne vas pas; mais quoi? Supposons qu'un élève nous pose la question. Attention, ne répondez pas à la légère ou par un argument d'autorité, car un élève qui pose une telle question pourrait bien être la réincarnation de STENDHAL (voir *"L'Ouvert"* n°23, fév. 81, page 2). Mais trouvez vous-même la réponse car je ne la fournirai pas ici.

Il est couramment admis que le système arabe de numération permet, lui, d'écrire n'importe quel nombre; bien entendu, il n'en est rien : il est aussi difficile d'écrire 10 puissance un milliard en chiffres arabes que un milliard en chiffres romains. C'est d'ailleurs la raison pour laquelle on a inventé l'écriture exponentielle $10^{1000000000}$. On pourra, certes, rétorquer que cette écriture exponentielle n'est qu'une variante, une extension évidente de l'écriture décimale; mais il n'est guère difficile de trouver un nombre trop grand pour pouvoir être écrit même sous forme exponentielle, et cela simplement en introduisant une nouvelle écriture, la notation de KNUTH : on pose $a \uparrow b = a^b$ puis $a \uparrow\uparrow b = a \uparrow (a \uparrow b), a \uparrow\uparrow\uparrow b = a \uparrow^3 b = a \uparrow\uparrow (a \uparrow\uparrow b)$ et ainsi de suite par récurrence $a \uparrow^n b = a \uparrow^{n-1} (a \uparrow^{n-1} b)$. Essayez maintenant d'écrire le nombre $10 \uparrow^{1000000000} 10$ en notation exponentielle! Pour donner une idée concrète de la puissance d'une telle notation, voyons les premiers exemples :

$$2 \uparrow 2 = 4$$
$$2 \uparrow\uparrow 2 = 16$$
$$\text{mais } 2 \uparrow\uparrow\uparrow 2 = 2^{65536} \simeq 2.10^{19728}$$

on peut montrer que

$$2 \uparrow^2 2 = 2^{2^2},\ 2 \uparrow^3 2 = 2^{2^{2^{2^2}}} \text{ (4 exposants successifs)}$$

et par récurrence :

$$2 \uparrow^n 2 = 2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}} \quad (2^{n-1} \text{exposants successifs}).$$

Ainsi pour écrire $2 \uparrow^{30} 2$ en notation exponentielle il faudrait (avec les caractères typographiques de cette page) une feuille de papier de 2 000 km de côté : les exposants successifs occuperaient toute la longueur de la diagonale. Pour écrire $2 \uparrow^{100} 2$, il faudrait une feuille de deux cent milliards d'années-lumière de côté!

Peut-on trouver des nombres qui ne peuvent pas être écrits dans la notation de KNUTH? Rien de plus facile, direz-vous, il suffit d'inventer une notation encore plus puissante. D'ailleurs, celle-ci existe déjà (cela s'appelle la procédure de GRAHAM) :

$$\begin{aligned} \text{on pose } a\Gamma\ b &= a \uparrow b \\ a\Gamma^2 b &= a \uparrow^{a\Gamma b}\ b \\ a\Gamma^3 b &= a \uparrow^{a\Gamma^2 b}\ b \\ \text{et par récurrence } a\Gamma^n b &= a \uparrow^{a\Gamma^{n-1} b}\ b. \end{aligned}$$

Rien ne nous empêche de continuer ainsi pendant des heures, si ce n'est peut-être une vague éthique qui nous fait soudain songer aux populations déshéritées du tiers-monde dont l'étalage de ces chiffres vertigineux, affublés d'unités telles que le dollar ou — encore plus désopilant — la mégatonne, ne peut qu'accroître la souffrance.

Toute cette multiplicité de procédés d'écriture nous donne le sentiment qu'en perfectionnant progressivement le langage numérique, nous pouvons couvrir des domaines de plus en plus vastes; il suffirait pour cela d'enrichir le langage de l'arithmétique, qui au départ ne comporte dans son alphabet que les 10 chiffres décimaux et les cinq symboles algébriques $+, -, \times,$: et $=$, en lui adjoignant successivement $\uparrow$, puis $\uparrow^n$, puis Γ^n. Or, en réalité, nous ne couvrons pratiquement rien ainsi. *Il se trouve qu'il y a une différence essentielle entre l'écriture décimale et les extensions que nous venons de présenter* : lorsqu'un nombre (par exemple mille milliards) est écrit en notation décimale, tous les nombres plus petits peuvent également être écrits, sans plus de peine puisqu'ils ne comportent pas plus de chiffres, mais ce n'est plus le cas pour un nombre tel que $10 \uparrow^{10} 10$. Ainsi, écrire 1 480 768 066 051 (nombre de 13 chiffres pris au hasard) n'est pas plus pénible que d'écrire 1 000 000 000 000 (nombre de 13 chiffres qui n'est pas pris au hasard). Par contre, un nombre tel que $10 \uparrow^3 10$, comporterait en écriture décimale $10^{10^{10^{10}}}$ chiffres; notez bien que si nous remplissions de chiffres les pages de *"L'Ouvert"* et si nous remplissions complètement, avec de tels numéros de *"L'Ouvert"* , un cube de un milliard d'années-lumière de côté, nous pourrions stocker 10^{84} chiffres environ. On peut donc soumettre au Directeur de l' I.R.E.M. un projet consistant à écrire un nombre de 10^{84} chiffres puisqu'il y a la place dans l'univers pour archiver les numéros de *"L'Ouvert"* que cette opération nécessite, mais vous voyez bien que pour un nombre de $10^{10^{10^{10}}}$ chiffres, ce ne serait pas réaliste. La notation de KNUTH permet d'écrire de très grands nombres, mais parmi les très grands nombres, **très peu peuvent être écrits en notation de KNUTH.** Cela montre que s'il est facile d'écrire (en notation de KNUTH) le nombre $10 \uparrow^4 10$, il est impossible d'écrire un

nombre "au hasard" de l'ordre de $10 \uparrow^3 10$, qui est pourtant bien plus petit. Mais qu'est-ce qu'un "nombre au hasard" ?

2.— Les nombres et les algorithmes.

Le concept de nombre au hasard (ou plus techniquement : nombre aléatoire) n'est pas évident; ainsi vous ne pouvez pas dire qu'un nombre est au hasard simplement du fait que ses chiffres binaires (par exemple) ont été tirés à pile ou face : il se pourrait en effet qu'en tirant $10 \uparrow^4 10$ fois de suite, vous obteniez par malchance 0 fois face et $10 \uparrow^4 10$ pile, vous donnant ainsi le nombre 0 qui n'est pas au hasard.

Fort heureusement ce concept a fait l'objet d'une étude absolument magistrale par Emile BOREL (1905). Emile BOREL dit qu'un nombre est aléatoire s'il ne peut pas être défini par un algorithme; plus exactement : un nombre est aléatoire s'il ne peut pas être défini par un algorithme plus simple que la simple donnée de ses décimales successives. Selon cette définition les nombres de base 0, 1, 2, ...9 sont forcément aléatoires (mais cela est peu intéressant). Par contre le nombre suivant : 123456789101112131415...(Vous écrivez tous les entiers naturels à la suite les uns des autres dans l'ordre croissant et sans laisser de blanc, jusqu'à l'entier un milliard) est non aléatoire car tout le monde conviendra que les instructions entre parenthèses, qui constituent l'algorithme, sont bien plus courtes que la donnée directe des 8 888 888 899 décimales du nombre. Toutefois, pour que cette définition soit effectivement opératoire, on doit pouvoir mesurer de façon précise la complexité d'un algorithme, afin de décider sans ambiguïté si oui ou non un algorithme donné est "plus simple que la donnée directe des décimales". Or aujourd'hui cela n'est plus un problème; les langages de programmation sont des langages impeccablement codifiés, et la complexité d'un programme sera (par définition) la quantité d'information qu'il contient, comptée en bits : c'est donc toujours un entier. La complexité ainsi définie dépend évidemment du langage utilisé, du microprocesseur, du système d'exploitation, etc. Mais sur une calculatrice donnée, avec un langage donné, elle est toujours bien définie (et affichée après compilation). Pour les besoins de cet article, un langage extrêmement simple suffit amplement, et pour fixer les idées, nous choisirons le langage de la T.I. 66. Avant de poursuivre, voici cependant une remarque importante :

> *Dans tout cet article, l'analyse que nous ferons des nombres, des algorithmes, ou des programmes fait totalement abstraction des limites techniques de la calculatrice, telles que les capacités de mémoires, le nombre de décimales affichées ou retenues, etc. Seul est pris en compte le langage lui-même, alphabet et syntaxe*

En adoptant ces hypothèses de travail, la complexité d'un algorithme (ou programme) est maintenant une grandeur parfaitement définie : c'est le nombre de pas du programme; bien entendu, quoique la T.I. 66 ne puisse enregistrer plus de

512 pas de programme, son langage permet néanmoins d'écrire des programmes de 1 000, 10 000, ou (si on en a la patience) 100 000 pas, et c'est bien cela qui nous intéresse. Si nous voulons désigner un nombre quelconque nous pouvons indiquer la liste de ses décimales : cela constitue bien un programme T.I., il suffit d'appuyer successivement sur les touches numériques correspondant à chacune de ces décimales. La complexité d'un tel programme est alors le nombre de décimales du nombre. Soit par exemple le nombre 10 000 000 000 (dix milliards); désigné sous cette forme décimale il comporte onze chiffres et le programme correspondant commande d'appuyer une fois sur la touche 1 puis dix fois sur la touche 0 : sa complexité est égale à 11. Mais nous pouvons écrire ce même nombre sous la forme 10^{10}, et, puisque nous avons une touche "puissance" (notée ici $\uparrow$) nous pouvons désigner le même nombre dix milliards par le programme $10 \uparrow 10$ qui comme vous le voyez vous-même, ne comporte que cinq pas (autrement dit, sa complexité est 5). Par conséquent, le nombre dix milliards n'est pas aléatoire, puisqu'il a pu être désigné par un programme plus simple (de complexité égale à 5) que la donnée directe de ses décimales (dont la complexité est 11). Si nous avions pris le nombre 10 460 353 203, il pourrait sembler à première vue qu'il est pris au hasard, mais ne vous y fiez pas : il est obtenu par le programme $3 \uparrow 21$ (de quatre pas). Lorsqu'un nombre est tiré au hasard,il est donc en général très difficile de s'assurer qu'il est aléatoire selon la définition d'Emile BOREL, mais peu importe car là n'est pas la question.

En plus du concept de nombre aléatoire, introduisons encore un autre concept qui clarifiera nos discussions ultérieures : *l'entropie d'un nombre.* C'est tout simplement le rapport :

$$\text{entropie de N} = \frac{\text{complexité du plus simple programme donnant N}}{\text{nombre de chiffres décimaux de N}}$$

Nous voyons immédiatement que l'entropie est toujours ≤ 1, et qu'elle est *égale* à 1 précisément pour les nombres aléatoires; les nombres aléatoires sont donc les nombres dont l'entropie est maximale. Bien sûr, si vous me donnez, comme ça, un nombre de cinquante chiffres "pris au hasard" (par exemple les résultats des matches de foot de ce week-end dans le Bas-Rhin) il me sera impossible de calculer son entropie : le dénominateur de la fraction vaut évidemment 50, mais le numérateur (sauf hasard miraculeux) sera absolument non-évaluable. Cela n'enlève cependant rien à l'intérêt de ce concept, comme nous ne tarderons pas à le voir.

L'entropie dépend du langage de programmation, mais cette dépendance est inessentielle : les valeurs exactes de l'entropie changeraient, certes, avec le langage de programmation, mais ces valeurs exactes sont presque toujours inconnaissables; tandis que les propriétés qualitatives (les seules à nous être utiles) sont pratiquement indépendantes de ce choix.

Nous voici maintenant mieux armés théoriquement pour poursuivre notre discussion.

Nous avons vu plus haut que si on peut écrire un milliard en notation décimale, il

n'est pas plus difficile d'écrire n'importe quel autre nombre à neuf chiffres, tandis que si on peut écrire $10 \uparrow^3 10$ en notation de KNUTH, on ne peut, notation de KNUTH ou pas, écrire n'importe quel nombre de cet ordre. Reprenons cela en termes d'entropie. Parmi les nombres à neuf chiffres, il en est (la plupart, d'ailleurs) qui sont aléatoires; mais quant aux autres, leur entropie ne peut guère devenir inférieure à 1/3, un cas optimum étant par exemple le nombre $387\,420\,489 = 9^9$: il peut être obtenu par le programme $9 \uparrow 9$ dont la complexité est 3, soit une entropie de $3/9 = 1/3$. Par contre l'entropie de $10 \uparrow^3 10$ est pratiquement nulle; en effet le programme suivant calcule $10 \uparrow^3 10$:

4
$x \sim t$
0
STO 1
10
STO 0
LBL 0
10
$\uparrow$
RCL 0
=
STO 0
1
STO + 1
RCL 1
INV 2nd $x \leq t$
GTO 0
RCL 0

ce programme comporte 20 pas, mais n'essayez pas de l'exécuter sur votre T.I. 66, qui n'est pas prévue pour afficher des nombres de cet ordre! Ce programme étant vraisemblablement le plus court possible, nous obtenons pour l'entropie de $10 \uparrow^3 10$:

$$\frac{20}{10^{10^{10^{10}}}}$$

c'est-à-dire un nombre si prodigieusement petit que cette fois nous n'aurions guère de scrupules à l'affubler de mégatonnes.

Nous pouvons donc conclure de ces considérations que, parmi les très grands nombres, ceux que nous pouvons désigner ont tous une entropie quasi-nulle. Par ailleurs, nous avons calculé tout à l'heure qu'il est matériellement impossible d'écrire des textes de plus de 10^{84} signes. Il est donc certain que tous les programmes que nous écrirons jamais auront une complexité inférieure à 10^{100} (vous voyez : je compte large). Le nombre de tous les programmes de complexité $\leq 10^{100}$ est donc inférieur au nombre total de textes pouvant être écrits avec les 25 touches de programmation et comportant au plus 10^{100} lettres, autrement dit

inférieur à $25^{10^{100}+1} - 1$. (En fait le nombre de programmes, quoique immense, est bien inférieur à cela, puisque très peu de ces textes respectent la syntaxe et possèdent le sens nécessaire pour constituer un programme; mais puisque nous comptons large ...) Cet argument nous prouve que parmi tous les nombres compris entre 0 et $10 \uparrow^3 10$, très peu sont programmables : la densité de ces derniers est en tous cas inférieure à

$$\frac{25^{10^{100}}+1}{10 \uparrow^3 10}$$

c'est-à-dire, là encore, quasi nulle. *Conclusion : presque tous les nombres sont non programmables* (synonyme ancien : non assignables).

A l'inverse, nous appellerons entiers assignables ou *entiers standard* tous les nombres qui peuvent être effectivement programmés, tandis que nous appellerons entiers accessibles ceux qui peuvent être effectivement écrits en écriture décimale. D'après ce que nous avons déjà dit plus haut (et qui est néanmoins vrai) tout entier inférieur à un entier accessible est accessible; mais $10 \uparrow^3 10$ est non accessible et par conséquent supérieur à tous les entiers accessibles. Pourtant, de même que "le plus grand nombre pouvant être écrit en chiffres romains", le plus grand nombre accessible n'existe pas.

3.— Que pouvons-nous savoir des très grands nombres?

Imaginez que je vous présente un nombre de 37 chiffres et que je vous demande : "est-il divisible par 7?" Vous n'aurez aucun mal à me répondre, puisqu'il vous suffira par exemple d'effectuer la division.

La situation n'est pas du tout la même pour les nombres inaccessibles. Je ne parle même pas des nombres qui ne sont ni accessibles ni programmables (notez en passant que accessible $\Rightarrow$ programmable) car ceux-là je ne peux même pas vous les donner, donc vous ne risquez pas de vous faire coller. Mais prenons un nombre programmable et non accessible, par exemple $10 \uparrow^3 10 - 1$. Je peux affirmer : "ce nombre est divisible par 9", mais pas question d'effectuer la division ou toute autre vérification directe. Le seul moyen en ma possession pour prouver une telle affirmation est de prouver d'abord par un raisonnement algébrique abstrait un théorème de la forme $\forall n P(n)$ et de dire ensuite "si $P(n)$ est vrai pour tout n, il est donc vrai aussi pour n inaccessible". Pour le nombre indiqué, je procéderai de la manière suivante :

1. Pour tout n, $10^n - 1$ est divisible par 9.
2. Or, $10 \uparrow^3 10 - 1$ est de la forme $10^n - 1$.
3. Donc, $10 \uparrow^3 10 - 1$ est divisible par 9.

Prouver un théorème de la forme $\forall n P(n)$ se ramène toujours en dernière analyse à une (ou plusieurs) démonstration(s) par récurrence; les prémisses qui sont à notre disposition pour déduire notre théorème sont évidemment toutes contenues dans les instructions du programme : nous ne savons rien d'autre sur le nombre que ce qui est dit dans le programme qui le définit. Ainsi, dans la mineure du syllogisme précédent, le seul renseignement que nous utilisions, à savoir que $10 \uparrow^3 10$ est

une puissance de 10, nous l'avons puisé dans la construction par exponentiations successives, c'est-à-dire dans le programme. Quant à la majeure, de la forme $\forall n P(n)$ ("tous les hommes sont mortels") elle se prouve par récurrence :

a) $10^1 - 1 = 9$ est divisible par 9,

b) supposons que $10^n - 1$ soit divisible par 9; alors $10^{n+1} - 1 = 10^{n+1} - 10^n + 10^n - 1 = 10^n \times 9 + 10^n - 1$ est aussi divisible par 9.

Nous venons de rencontrer la loi fondamentale de l'arithmétique des nombres inaccessibles :

> "Tout ce que nous pouvons savoir sur un nombre est obtenu par une récurrence, suivie d'un syllogisme sur le modèle ci-dessus."

De façon plus précise : la récurrence sert à prouver la majeure du syllogisme, et la mineure se déduit des informations contenues dans le programme. Il n'existe aucun autre moyen de connaître quoi que ce soit sur un nombre inaccessible.

4. Et les nombres décimaux?

Un nombre décimal n'est, au fond, rien d'autre qu'un nombre entier accompagné d'une indication d'échelle : la virgule. Lorsque nous additionnons, multiplions, ou divisons des nombres décimaux, nous faisons la même chose que pour des entiers, avec juste quelques règles supplémentaires concernant la place de la virgule. Et d'ailleurs, lorsque nous divisons deux entiers, nous obtenons un quotient qui est entier ou décimal selon que nous retenons ou pas les chiffres après la virgule.

Tout ce que nous avons dit sur les entiers, la complexité des algorithmes, l'entropie, les nombres aléatoires, etc, marche aussi bien pour les nombres décimaux, avec toutefois quelques détails nouveaux dont nous allons parler.

La principale différence concerne le nombre de chiffres : lorsque nous indiquons un nombre entier en donnant un programme, le nombre de chiffres du nombre est également donné : le nombre de chiffres peut être un nombre compliqué ou difficile à calculer, mais il est implicitement contenu dans le programme,sans aucune ambiguïté. Au contraire, si par exemple nous effectuons la division 3 : 7, nous n'obtiendrons jamais un reste nul, donc nous ne rencontrerons jamais une indication d'arrêt. Une telle indication d'arrêt doit être ajoutée au programme : on peut y mettre une instruction signifiant "arrêtez-vous au 12^{e} chiffre après la virgule", ou bien "arrêtez-vous au 3 588 232 121 441 862 657^{e} chiffre après la virgule", ou encore "arrêtez-vous au $10 \uparrow^3 10^{\text{e}}$ chiffre". On remarquera que pour pouvoir parler du nombre de chiffres d'un nombre décimal il faut une telle indication d'arrêt; à cette condition seulement, nous pourrons définir l'entropie d'un nombre décimal comme nous l'avions fait pour les nombres entiers. Sans indication d'arrêt, on ne parlera donc pas de l'entropie, mais on pourra néanmoins mesurer la complexité d'un nombre décimal par la complexité du plus court programme qui le définit; en outre, cette dernière considération, qui est quantitative, induit automatiquement la distinction qualitative que voici : comme pour les entiers, on peut constater que

parmi les nombres décimaux certains sont plus égaux que les autres; il y a l'élite (les nombres non aléatoires) et la masse anonyme des nombres aléatoires. Comme il se doit, l'élite est peu nombreuse; en effet, le raisonnement que nous avons tenu plus haut à propos des nombres entiers en majorant l'entropie peut être reproduit pour les nombres décimaux tronqués à un ordre n (tronqués, c'est-à-dire qu'on ne retient que les n premières décimales en oubliant les autres); on voit alors que quel que soit l'ordre n de la troncature, pourvu qu'il soit grand, les nombres non aléatoires sont rares.

La définition de l'aléatoire selon Emile BOREL est un outil conceptuel remarquable, dont on n'a pas fini d'assimiler le mode d'emploi. Toutefois, bien que très opératoire sur le plan théorique, elle ne décrit pas exactement l'aléatoire tel qu'on le rencontre en pratique. Voyons cela de plus près.

Supposons un nombre comportant beaucoup de chiffres, de quoi remplir par exemple une page de "*L'Ouvert*", soit deux mille chiffres environ. Il se peut que ce nombre puisse être déterminé par un programme de mille cinq cent pas, mais par aucun programme plus court. De tels nombres, il y en a assurément plein, comme on peut s'en convaincre facilement avec un argument de dénombrement analogue à celui qui nous avait servi à prouver la rareté des nombres non aléatoires ([1]). Un tel nombre est non aléatoire si on s'en tient à *la lettre* de la définition; mais il y a aussi l'esprit de la définition : BOREL nous a légué une certaine vision du hasard et de la chose numérique qui d'une manière ou d'une autre va à l'encontre d'une lecture trop étroite ou trop formaliste. Ainsi l'esprit de cette vision du hasard pourrait s'exprimer par la traduction suivante : "*le hasard, c'est ce qui n'est déterminé par aucun algorithme accessible*". La définition de BOREL, lue à la lettre, est un artifice théorique commode pour l'analyse mathématique (par exemple pour démontrer des théorèmes), mais il est visiblement peu naturel de tenir pour remarquable, c'est-à-dire non aléatoire, un nombre comme celui qui vient d'être envisagé. Autrement dit, les nombres remarquables sont les nombres déterminés par des programmes simples. Le mot "simple" ayant un sens trop subjectif, on comprend que BOREL ait créé de toutes pièces la définition que nous discutons. Dans le langage de la mathématique non-standard, on dirait que la définition "*un nombre est aléatoire s'il ne peut être calculé par un programme plus court que la donnée directe de ses décimales*" est interne, tandis que la définition "*un nombre est aléatoire s'il ne peut être calculé par aucun programme accessible*" est externe. Mais bien entendu, **elles ne sont pas équivalentes!**

Il ressort de ces considérations que la définition suivante, interne elle aussi, est en tout cas très pertinente :

Définition .— *On appelle complexité (ou degré de complexité) d'un nombre la complexité du plus court programme qui le calcule.*

([1]) Evidemment, il est difficile d'en exhiber un avec la preuve qu'il ne peut être défini par un programme plus court. Si un programme connu est réputé le plus court connu à ce jour, en trouver un plus court est déjà une gageure. Mais montrer *qu'il n'en existe pas* de plus court...

On évitera, pour une raison que tout lecteur de *"L'Ouvert"* comprendra spontanément, d'appeler nombre complexe un nombre dont le degré de complexité est élevé; on dira plutôt : nombre compliqué, ou nombre non-standard. On évitera également de l'appeler nombre aléatoire, malgré ce qui a été dit ci-dessus; il vaut mieux réserver cette appellation pour les nombres qui répondent à la définition *interne*. A l'inverse, les nombres dont le degré de complexité est bas pourront être appelés nombres *simples* ou (mieux) *remarquables*, ou *standard*. Tout cela s'applique d'ailleurs indifféremment aux nombres entiers ou aux nombres décimaux.

5.— Les nombres vus en perspective.

Emile BOREL nous a ainsi montré que les nombres ne sont pas simplement des choses en soi que le mathématicien étudie : les connaissances que nous pouvons acquérir sur eux ne dépendent pas seulement de leurs propriétés intrinsèques (que je pourrais appeler objectives), mais aussi de la perception que nous en avons, de par nos limites intellectuelles à nous, les observateurs. De même que nous connaissons mieux la géologie de la Terre que celle de Vénus, de même que nous connaissons mieux la morphologie du Soleil que celle d'Alpha du Centaure, nous connaissons mieux les nombres simples que les nombres compliqués. Nous ne voyons pas tous les nombres : ceux que nous voyons sont les nombres standard ou assignables, mais comme il a été démontré plus haut, cela fait très peu : l'immense majorité sont des nombres que nous ne voyons pas ([2]). La mathématique héritée des anciens grecs classait les nombres selon une nature : entier, rationnel, irrationnel, comme l'astronomie disposait les astres sur une sphère. La mathématique à l'ère de l'ordinateur (conformément à la vision du génial précurseur BOREL) perçoit les nombres dans une perspective où la distance est le degré de complexité, tout comme l'astronomie à l'ère du télescope a révélé que le firmament avait une profondeur et que la majorité des astres sont trop obscurs et éloignés pour être visibles.

Je ne discuterai pas le problème de savoir si cette perspective sous laquelle nous voyons les nombres est objective ou absolue, ou au contraire due seulement à notre manière de concevoir les nombres, car ce problème est trop métaphysique. (Il cesserait d'être métaphysique si on découvrait une arithmétique autre que celle que nous connaissons et réellement efficiente, mais qui modifierait complètement cette perspective; comme nous ne connaissons pas une telle arithmétique, nous ne pouvons que spéculer sur son existence, ce qui est assez stérile.) Par contre, il est instructif d'examiner quelques propriétés évidentes de cette perspective, particulièrement celle que voici.

Appelons $c(x)$ la complexité du nombre x. Lorsqu'on effectue entre deux nombres x et y une opération de l'arithmétique $+$, $-$, $\times$ ou $\cdot/\cdot$ on en obtient un troisième,

([2]) Un sophisme répandu (mais ceux qui le professent n'y croient évidemment pas plus que tout un chacun) affirme que seuls existent les nombres que l'on peut définir, c'est-à-dire ceux "qu'on voit". Cependant, si on admet ce sophisme on doit admettre que entre les nombres $10 \uparrow^3 10$ et $10 \uparrow^4 10$ il y a plein de trous, ce qui conduit à abandonner l'axiome de récurrence. Or, je n'ai jamais rencontré un adepte de ce sophisme qui récuse — en conséquence — l'axiome de récurrence. Pour être conséquent le sophiste doit professer qu'aucun nombre n'existe (mais curieusement ces sophistes savent combien ils gagnent).

noté $x * y$, où $*$ désigne génériquement l'un des quatre symboles précédents. Or, il est bien clair que si on a un programme qui calcule x et un autre qui calcule y, on obtient un programme qui calcule $x * y$ en juxtaposant simplement les deux premiers ([3]). Ce qui conduit à l'inégalité :

$$C(x * y) \leq c(x) + c(y).$$

En outre, chacune des quatre opérations a son inverse parmi les quatre, ce qui permet d'écrire aussi les inégalités :

$$c(x) \leq c(y) + c(x * y)$$
$$c(y) \leq c(x) + c(x * y)$$

Comme tout professeur de mathématiques le sait, ces trois inégalités signifient qu'on peut faire un triangle avec trois segments de longueurs $c(x), c(y)$ et $c(x * y)$; ce qui peut se résumer aussi par les inégalités :

$$|c(x) - c(y)| \leq c(x * y) \leq c(x) + c(y).$$

Pour chacune des quatre valeurs possibles du signe $*$ on peut en déduire des propriétés de l'opération arithmétique correspondante; comme je ne peux pas tout raconter ici, je dirai juste quelques mots au sujet de l'opération $-$, mais rien ne vous empêche de chercher vous-même autre chose.

A vrai dire, ce que nous pouvons trouver ainsi, ce sont des propriétés de l'opération non en elle-même, mais relativement à la perspective que donne la complexité.

Soit donc x un nombre standard (c'est-à-dire dont la complexité est faible) et supposons qu'on veuille approcher x par des nombres y successifs; on aura toujours l'inégalité

$$c(y) \geq c(x - y) - c(x).$$

Or, on ne peut atteindre des valeurs de plus en plus petites de $x - y$ (c'est-à-dire se rapprocher indéfiniment de x) sans que la complexité de $x - y$ croisse au-delà de toute limite : quoique je ne l'aie pas encore dit pour les petits nombres (mais je l'ai dit au début pour les entiers grands), vous l'avez maintenant bien compris. L'inégalité ci-dessus nous dit alors que la complexité de y va elle aussi croître au delà de toute limite. Il est donc impossible d'approcher un nombre simple par d'autres nombres simples. Ou encore, si vous préférez : un nombre simple

([3]) Je néglige là quelques petits détails peu importants. En toute exactitude, la juxtaposition exige quelques pas de programme en plus, à titre d'interface. Par exemple si le programme P calcule x et si Q calcule y on obtient un programme R qui calcule $x + y$ en enregistrant P, puis $STO0$, puis Q, suivi de $+RCL0$ (ce qui fait cinq pas d'interface). On peut toujours dire que ces pas d'interface sont si peu nombreux qu'on peut les négliger (surtout si P et Q sont longs). On peut dire aussi qu'en juxtaposant deux programmes il y a toujours moyen de mettre en commun cinq instructions, ce qui compense l'interface.

(standard) est toujours isolé dans un halo de nombres infiniment proches de lui mais tous inassignables.

La grosseur de ce halo dépend évidemment de l'intensité du mot "inassignable", c'est-à-dire en dernière instance de la richesse du langage de programmation qui sert de référence. Ainsi, si on peut programmer la procédure de GRAHAM dont il a déjà été question, ces halos seront d'une petitesse inimaginable (et sans la moindre incidence pratique). Par contre, avec des langages très rudimentaires une grosse exponentiation comme $10 \uparrow\uparrow 10$ est déjà inaccessible (n'oublions pas que le langage de référence n'est pas forcément le langage de programmation qu'on utilise pour travailler : on peut programmer en PASCAL tout en mesurant la complexité avec un langage extrêmement rudimentaire) et alors les halos pourront être vus ([4]).

De toute façon l'inégalité que nous étudions peut donner des renseignements plus quantitatifs : soit par exemple le nombre a calculé par le premier programme donné en annexe : $c(a) \leq 29$; soit maintenant ϵ le plus petit nombre pouvant être calculé par un programme de 100 pas (ou moins). Par définition de ϵ, si $|a-y| \leq \epsilon$, alors $c(a-y) \geq 100$, donc d'après notre inégalité $c(y) \geq c(a-y) - c(a) \geq 100 - 29 = 71$. Ce qui veut dire que tous les nombres compris entre $a-\epsilon$ et $a+\epsilon$ et différents de a ont une complexité supérieure à 71. De même si ϵ' est le plus petit nombre programmable en moins de 1 000 pas, alors tous les nombres compris entre $a-\epsilon'$ et $a+\epsilon'$ (et distincts de a) ont une complexité supérieure à 971. Bien sûr, ϵ et ϵ' sont des nombres très petits : leur définition même montre qu'ils sont très peu aléatoires, donc bien plus petits que 10^{-100} (pour ϵ)ou 10^{-1000} (pour ϵ').

Une dernière remarque pour en finir avec la Perspective : supposons que notre langage de référence soit le plus pauvre de tous; dépourvu de toutes les opérations arithmétiques, il se réduirait aux dix touches numériques 0,1,2,3,4,5,6,7,8,9. Dans ce cas (dégénéré) la complexité d'un nombre serait simplement le nombre de ses décimales; en particulier, les nombres décimaux assignables seraient nécessairement tronqués et par exemple les nombres a ou b de l'annexe seraient des nombres inaccessibles. Tous les nombres seraient aléatoires et la Perspective complètement aplatie. Autrement dit, ce qui fait que les nombres a ou b de l'annexe sont remarquables, c'est l'existence des opérations de l'arithmétique.

6.— La vieille controverse sur le continu n'était pas épuisée.

Il n'y a pas que les nombres; il y a aussi les fonctions; et les fonctions aussi

([4]) Je ne veux pas entrer ici dans une discussion détaillée sur la mesure de la complexité : il faudrait un livre. On s'en tiendra donc aux conventions retenues pour cet article : choix d'un langage pour fixer les idées. Il peut cependant être utile ici de préciser ce qui suit. Supposons que nous cherchions à programmer la procédure de GRAHAM sur T.I. 66; on devrait commencer par traduire la construction récursive de KNUTH, ce qui est possible en imbriquant des sous-programmes (touche SBR ou avec GTO). Alors le calcul de $a \uparrow^n b$ appellerait le sous-programme qui calcule $a \uparrow^{n-1} b$; il faudrait donc n sous-programmes pour calculer $a \uparrow^n b$. On voit ainsi pourquoi la procédure de GRAHAM ne peut pas être programmée, même de manière purement syntactique : elle pourrait l'être si on disposait de plusieurs touches $x \sim t$ indépendantes. Un problème analogue se pose si on veut programmer des puissances en itérant des additions (mais sa difficulté est bien moins radicale).

peuvent être calculées par des programmes. De même que pour les nombres, on peut distinguer les fonctions programmables ou standard, et les fonctions non programmables. Un programme qui calcule une fonction diffère des programmes qui calculent un nombre uniquement en ce qu'il appelle une mémoire libre, c'est-à-dire une mémoire que l'utilisateur devra charger avant l'exécution du programme (remarquez que dans les programmes A et B de l'annexe il n'y a pas de mémoire libre : toutes les mémoires appelées ont été chargées par des instructions du programme); cette mémoire libre reçoit la valeur de la variable, et le programme calcule alors la valeur correspondante de la fonction. Pour une fonction de n variables, il y aurait n mémoires libres. La complexité $c(f)$ d'une fonction f sera alors la longueur du plus court programme qui la calcule. Si x est un nombre non assignable rien n'empêche de *parler* de $f(x)$, mais pas question de le *calculer* même si $c(f) = 5$: si f est standard et x non standard, alors $f(x)$ est forcément non standard : en effet, si $y = f(x)$ était standard, on pourrait construire un programme résolvant l'équation $f(x) = y$, donc x serait programmable. De façon plus précise, on peut écrire des inégalités analogues à celles du paragraphe précédent :

$$c(f(x)) \leq c(f) + c(x).$$

Autrement dit : les lois de la Perspective s'étendent aux fonctions. Je ne peux m'étendre plus longuement sur les fonctions, mais je voudrais finir en reparlant d'une vieille chose : l'infini selon CANTOR et DEDEKIND. Comme vous le savez (vous avez dû apprendre cela à l'université), DEDEKIND a prétendu donner la première définition mathématique rigoureuse de l'infini, et pour cela il a fait appel à la notion de fonction, plus fondamentale selon lui. Son idée était simplement qu'un ensemble est infini si (et seulement si) il existe une *bijection* entre cet ensemble et une de ses parties. Or, considérons l'ensemble des entiers entre 0 et $10 \uparrow^4 10$:

$$E = \{0, 1, \ldots 10 \uparrow^4 10\}.$$

Nous avons vu au paragraphe 2 que presque tous les éléments de E étaient non programmables. Soit alors l'application f de E dans E ainsi définie :

$$f(n) = \begin{cases} 2n \text{ si } n \text{ est programmable} \\ n \text{ si } n \text{ est non programmable.} \end{cases}$$

Cette application est injective : en effet, soient n et p deux éléments de E tels que $f(n) = f(p)$:

– si n et p sont tous deux programmables, $2n = 2p$, donc $n = p$;

– si n et p sont tous deux non programmables, $n = p$;

– le cas où n serait programmable et p non programmable, ou l'inverse, ne peut pas se produire, car si n est programmable, $2n$ l'est également et vice versa.

Donc f est injective.

Mais f n'est pas surjective, car 3, 5 ou 73 ne sont pas des valeurs que f peut prendre.

Il n'y a donc que deux conclusions possibles : ou bien E est, selon DEDEKIND, un ensemble infini (ce qui n'est choquant que si on a acquis trop de certitudes), ou bien f n'est pas une application (idem). Peut-être penchez-vous déjà, spontanément, en faveur de la seconde solution, mais réfléchissez bien avant de dire une sottise : vous ne trouverez chez DEDEKIND aucune définition qui exclue f de la confrérie des applications; si on est porté à l'exclure, c'est en vertu d'un pressentiment, non d'un argument dûment éprouvé. D'ailleurs pour achever de vous convaincre que cette exclusion n'a rien d'une évidence, je pourrais vous dire que dans la mathématique non standard, qui fournit un modèle axiomatique du phénomène que nous sommes en train de discuter, on envisage expressément des applications telles que f (elles sont dites *externes*) et ce sont bien de "véritables" applications au sens le plus orthodoxe de la théorie des ensembles.

Bien sûr, la fonction f ci-dessus est non programmable (on ne peut pas traduire sa définition en algorithme, car elle comporte un test d'assignabilité), mais DEDEKIND n'a jamais dit que les vraies applications sont les applications programmables : cela, on en est certain, car POINCARÉ pensait, lui, que les vraies applications sont les applications programmables, et il y a eu une polémique à ce sujet entre lui et la bande à CANTOR-DEDEKIND. Visiblement DEDEKIND n'avait pas songé à ce paradoxe; s'il y avait été confronté, par un contradicteur par exemple, il aurait probablement cherché à affiner le concept de fonction afin d'exclure f. Mais peut-être aurait-il finalement préféré considérer que E est un ensemble infini.

Vous voyez donc que l'univers des nombres et des fonctions n'était pas aussi bien connu qu'on a pu le croire. CANTOR, DEDEKIND et HILBERT n'avaient pas remarqué cette Perspective qui n'est devenue sensible qu'avec l'ordinateur.

Plusieurs théories sont apparues et se sont fortement développées ces derniers temps, qui abordent le monde numérique dans l'esprit que j'ai essayé de vous faire partager. Ce sont notamment : la théorie des fonctions calculables, l'analyse non-standard, et la théorie de la complexité. On peut être sûr que sans l'avènement de l'ordinateur elles seraient restées des curiosités vite oubliées.

ANNEXE

Voici deux programmes que vous pouvez exécuter sur une calculatrice programmable Texas Instruments (ils sont exécutables tels quels sur les 57 II, 62, 66).

PROGRAMME A

0
STO 0
2
STO 1
4
÷
3
=
STO 2
1
STO 3
LBL 0
RCL 1
+
RCL 2
=
÷
RCL 3
=
STO + 0
4
+/−
STO ÷1
9
+/−
STO ÷2
2
STO + 3
GTO 0

(29 pas; 4 mémoires vives)

Ce programme calcule un nombre a dont la complexité est donc ≤ 29.

PROGRAMME B

1
STO 0
STO 1
STO 2
LBL 0
RCL 2
STO ÷1
RCL 1
STO + 0
1
STO + 2
GTO O

(12 pas; 3 mémoires vives)

Ce programme calcule un nombre b dont la complexité est donc ≤ 12.

Remarques

1.— Ces programmes ne comportent aucune instruction d'arrêt; vous devrez donc les arrêter manuellement au bout de quelques secondes, et vous lirez alors la valeur du nombre calculé (a ou b) dans la mémoire 0. Rien ne vous empêche d'ajouter de telles instructions pour vous amuser, mais il y a, relativement à la discussion développée dans l'article, un enjeu théorique. En effet, les nombres a et b sont irrationnels et ont donc une suite infinie (et pas du tout aléatoire) de décimales : si vous ajoutez des instructions d'arrêt, le programme ne calculera plus le même

nombre, mais un autre nombre proche (même si la différence entre les deux n'affecte pas les huit chiffres affichés). Par ailleurs, conformément à ce qui a été dit au § 5, ajouter des instructions d'arrêt allonge le programme; les approximations de a ou b seront des nombres *rationnels*, contrairement aux nombres a et b eux-mêmes, mais leur complexité sera plus élevée : elle sera même grande si vous voulez que l'erreur soit extrêmement petite, ou inaccessible si vous voulez que l'erreur soit plus petite que tout ce qui est accessible (voir les inégalités du § 5).

PROGRAMME A'

4
STO 0
1
STO 1
LBL 0
2
STO + 1
1
-
RCL 1
x^2
$1 : x$
=
STO × 0
GTO 0

(15 pas; 2 mémoires vives)

2.— Le programme B est vraisemblablement le plus court qui calcule b (mais on ne sait jamais ...). Par contre, il y a des programmes plus courts que A pour calculer a, par exemple celui qui est listé ci-dessus : la complexité de a est donc 15 (peut-être 14 ou 13, mais je crois qu'il est difficile de faire moins) car dans la définition de la complexité nous disons simplement "la longueur du plus court programme" sans spécifier aucune condition sur ce programme; or le programme A', quoique deux fois plus court, converge infiniment plus lentement que A : pour avoir huit chiffres exacts, nous devons répéter près de 80 000 000 fois la boucle de A', alors qu'il suffit de répéter 13 fois celle de A. Cela inclinera peut-être le lecteur à penser que la notion de complexité retenue ici manque de pertinence : un nombre peut-il être considéré comme simple si le seul programme simple qui le calcule est aussi lent que A' et que les programmes rapides sont tous longs? Cette critique est extrêmement juste, mais je répondrai que tout dépend de ce que vous voulez analyser avec votre complexité; selon le cas une définition incluant la rapidité peut être préférable.