DUPLICAAT

STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

DR 22r

L'influence de Algol 60 sur l'analyse numerique.

(Colloque sur l'analyse numerique,
C.B.R.M.,Louvain 1961,p 89-97).

A.van Wijngaarden.

1961

BIBLIOTHEEK MATHEMATISCH CENTRUM
AMSTERDAM

22

CENTRE BELGE DE RECHERCHES MATHÉMATIQUES

Extrait du

COLLOQUE

SUR

L'ANALYSE NUMÉRIQUE

TENU A MONS

LES 22, 23 ET 24 MARS 1961

CBRM

LIBRAIRIE UNIVERSITAIRE
10, RUE DE LA MONNAIE
LOUVAIN-BELGIQUE

LIBRAIRIE GAUTHIER-VILLARS
55, QUAI DES GRANDS-AUGUSTINS
PARIS

1961

L'INFLUENCE DE ALGOL 60 SUR L'ANALYSE NUMÉRIQUE

PAR

M. le Prof. A. VAN WIJNGAARDEN (Amsterdam) *)

Parce que le but du langage algorithmique ALGOL 60 [1] est la description des processus de calcul, il va sans dire que sa structure est influencée par l'analyse numérique. Parce que le but de ALGOL 60 n'est pas la prescription des processus de calcul il ne va pas sans dire qu'inversement l'analyse numérique sera influencée par ALGOL 60. Néanmoins ce sera le cas.

Dans le Mathematisch Centrum à Amsterdam il est en opération depuis juin 1960 un traducteur ALGOL 60 pour un des calculateurs électroniques de l'institut, du type Electrologica *X*1.

Ce traducteur [2] qui traduit pratiquement le langage complet, fait par MM. Dijkstra et Zonneveld, est un outil formidable pour le programmeur et directement après sa mise en œuvre on se trouve en face du problème d'écrire un système intégré des procédures standards pour les processus les plus communs dans l'analyse numérique. De propos délibéré on a dit «système intégré».

En fait il n'est pas trop difficile de décrire quelques processus séparés, mais il est difficile de les décrire ainsi qu'ils forment un complet logique qui utilise les mêmes idées de base et qui permet que les procédures utilisent l'une l'autre. Naturellement ça vaut plus ou moins pour chaque librairie des subroutines, mais par la généralité énorme de ALGOL 60 il y a besoin des procédures très générales, qui ensemble forment plus ou moins un livre sur l'analyse numérique. Et là on trouve que dans l'analyse numérique on traite souvent nonchalamment les choses très importantes, toutes reliées au manque d'exactitude qui établit les différences entre l'analyse numérique et l'analyse pure.

*) Mathematisch Centrum, Amsterdam.

Quand on décrit des processus en ALGOL 60, on doit choisir et il ne reste pas la possibilité de ne se prononcer sur les petits détails, parce qu'on ne peut pas dire en ALGOL 60 des choses comme «un nombre suffisant des termes», «jusqu'à la différence entre deux valeurs successives est suffisamment petite», «une valeur choisie convenablement», «avec une précision suffisante», etc.

Alors les manufacteurs des procédures ne récrivent pas seulement des chapitres de l'analyse numérique, mais aussi ils les écrivent d'une manière nouvelle. Leurs contributions peuvent être substantielles parce que les décisions additionnelles peuvent être d'une grande importance pour l'efficacité d'un certain processus. En outre ils ne peuvent pas expecter que les notions nouvelles qu'ils introduisent, soient acceptées par tous leurs collègues.

Déjà dans notre groupe il existe des opinions contraires sur les principes.

Quelques exemples peuvent clarifier ces idées.

La forme dans laquelle on décrit un processus en ALGOL 60 est celle d'une procédure. Commençons par définir quelques notions élémentaires, c.-à-d. la valeur absolue d'un nombre

real procedure *abs* (x); **value** x; **real** x;

$$abs := \textbf{if } x \geqslant 0 \textbf{ then } x \textbf{ else } -x,$$

le signe d'un nombre

integer procedure *sign* (x); **value** x; x **real**;

$$sign := \textbf{if } x > 0 \textbf{ then } 1 \textbf{ else if } x = 0 \textbf{ then } 0 \textbf{ else } -1,$$

le plus grand de deux nombres

real procedure *max* (x,y); **value** x,y; **real** x,y;

$$max := \textit{if } x \geqslant y \textbf{ then } x \textbf{ else } y,$$

et alors la plus grande des valeurs absolues de deux nombres

real procedure *maxabs* (x,y); **value** x,y; **real** x,y;

$$maxabs := max\,(abs\,(x), abs\,(y)).$$

Il commence d'être plus intéressant dès que la notion de la précision joue un rôle. Le rapport ALGOL 60 dit explicitement quelque chose sur se sujet :

3.3.6. Arithmetics of **real** quantities. Numbers and variables of type **real** must be interpreted in the sense of numerical analysis, i.e. as entities defined inherently with only a finite accuracy.

Similarly, the possibility of the occurrence of a finite deviation from the mathematically defined result in any arithmetic expression is explicitly understood. No exact arithmetic will be specified, however, and it is indeed understood that different hardware representations may evaluate arithmetic expressions differently. The control of the possible consequences of such differences must be carried out by the methods of numerical analysis. This control must be considered a part of the process to be described, and will therefore be expressed in terms of the language itself.

Le mathématicien numérique doit créer alors sous la forme de procédures les moyens qui le font possible de contrôler actuellement la précision. Nous définions premièrement la notion forme d'apparition d'un nombre réel.

Le calculateur a à sa disposition comme représentation des nombres réels une suite dénombrable des nombres réels ou bien des suites de symboles, qui chaque pour soi sont l'image d'un nombre réel et alors sont identiques à ces nombres réels. Cette suite est ordonnée car nous supposons que le calculateur est capable de discerner que deux nombres sont identiques ou non et quand ils ne le sont pas de trouver lequel est le plus grand. Naturellement tous les nombres réels ne peuvent pas avoir des représentations différentes et alors les opérations numériques comme l'addition, la multiplication et l'affectation ne peuvent pas être exécutées exactement en général.

Même dans le cas où une exécution exacte serait possible, il n'est pas supposé que le résultat soit exact.

Toutefois il est supposé que le résultat soit bien déterminé et que les règles soient ainsi que certaines relations d'ordre sont préservées, comme $x > 0 \supset x + y \geqslant y \wedge y + x \geqslant y$, *abs* (*sign* $(x \times y) - sign(x) \times sign(y)) \leqslant 1$.

L'idée est que l'ordre des nombres, qui est la seule notion qui reste exacte, ne soit pas bouleversé. Aussi les identificateurs qui apparaissent dans les expressions sont pour le calculateur les noms des formes d'apparition des nombres réels : c.-à-d. les valeurs qu'on obtient effectivement dans le calcul et pas nécessairement les noms des nombres réels eux-mêmes, c.-à-d. les valeurs qu'on aurait obtenu quand on aurait exécuté le calcul avec une précision infinie.

Alors une relation $x = y$ où x et y sont de type **real** ne présente

aucune difficulté, parce que les formes d'apparition de x et de y sont à chaque moment exactement connues et bien ils sont identiques ou bien ils ne le sont pas.

De l'autre côté on doit noter que dans deux exécutions du même programme, mais faites avec une autre arithmétique, disons sur un autre calculateur, la valeur de cette relation $x = y$ peut différer. Le mathématicien de son côté n'est pas intéressé naturellement dans les formes d'apparition, mais il doit accepter que le calculateur interprête ses identificateurs des nombres comme les noms des formes d'apparition de ces nombres. Alors il doit tenir compte du fait qu'au lieu d'un nombre x qui reste inconnu sa forme d'apparition x, qui peut varier selon le calculateur, mais qui est toujours connu, est pris.

Il peut alors définir que tout x satisfaisant $X - \mathit{deltamin}\ (X) \leqslant x \leqslant X + \mathit{deltaplus}\ (X)$ est une forme d'apparition acceptable de X et il peut définir la précision en choisissant les fonctions *deltamin* et *deltaplus*.

De nouveau il y a une restriction dans ce choix qui concerne l'ordre des nombres. Il semble naturel d'exiger que l'ensemble des nombres X dont x est une forme d'apparition acceptable est convexe, c.-à-d. que quand x est forme acceptable de $X1$ et de $X2 < X1$ il l'est aussi de chaque nombre X satisfaisant $X1 \leqslant X \leqslant X2$.

Aussi il semble naturel d'exiger que X lui-même est toujours une forme acceptable de X (alors qu'il est permis de ne pas faire d'erreurs). Cela veut dire que :

$$X1 < X2 \supset X1 - \mathit{deltamin}\ (X1) \leqslant X2 - \mathit{deltamin}\ (X2)$$
$$\wedge\ X1 + \mathit{deltaplus}\ (X1) \leqslant X2 + \mathit{deltaplus}\ (X2)$$

Un choix convenable est fait par l'introduction des concepts erreur relative *re* et erreur absolue *ae*, où $re < 1$ et $\mathit{deltamin} = \mathit{deltaplus} = \mathit{delta}$, défini par

real procedure *delta* (*x*, *re*, *ae*); **real** (*x*, *re*, *ae*);
delta : = *maxabs* ($x \times re$, *ae*).

Comme on le vérifie facilement ce choix satisfait les conditions mentionées ci-dessus et en outre est conforme aux notions usuelles. Quand on admet seulement une erreur relative on pose $ae = 0$ et quand on admet seulement une erreur absolue on pose $re = 0$,

pendant que le cas que le nombre soit exact est donné par $ae=re=0$.

Parce qu'on peut opérer seulement avec les formes d'apparition, les seuls nombres qu'on connaît, il n'est pas encore directement clair comment on peut appliquer le concept *delta*. Alors nous posons la question si deux nombres desquels on a les formes d'apparition x et y son différents ou égaux. Logiquement on peut seulement dire que deux choses sont égales quand on ne peut démontrer aucune différence entre toutes les propriétés observables.

La notion de différence est alors plus fondamentale et on peut seulement démontrer que x et y sont différents quand il n'existe pas un nombre duquel ils sont formes d'apparition acceptables tous les deux. Toutefois quand x et y ne sont pas formes acceptables de $m = (x + y)/2$ ils ne le sont d'aucun nombre. En effet soit $x < y$. Alors

$$x < m - delta\,(m) < m + delta\,(m) < y.$$

Quand $z > m$ on a

$$x < m - delta\,(m) < z - delta\,(z)$$

et quand $z < m$ on a

$$z + delta\,(z) < m + delta\,(m) < y.$$

Alors on est seulement tenu à considérer m et on a directement

```
Boolean procedure different (x, y, re, ae); value x, y;
                    real x, y, re, ae;
                    different: = delta ((x + y)/2, re, ae) <
                                 abs ((x - y)/2).
```

Quand deux nombres ne sont pas différents on les proclame égaux, à défaut de preuve pour ainsi dire, alors

```
Boolean procedure equal (x, y, re, ae);
                    real x, y, re, ae;
                    equal: = ¬ different (x, y, re, ae).
```

Nous donnons cette définition recursivement, mais naturellement rien ne s'oppose d'appliquer un peu d'algèbre de procédures, lequel est extrêmement difficile en général, mais trivial ici. Par conséquence on a la forme équivalente

```
Boolean procedure equal (x, y, re, ae); value x, y;
                    real x, y, re, ae;
                    equal :=delta((x+y)/2, re,ae) ⩾ abs((x−y)/2).
```

Naturellement le concept *equal* n'enjouit pas la propriété de transitivité. Alors on peut se demander si x est plus petit ou plus grand que y et on définit

Boolean procedure *smaller* (x, y, *re*, *ae*); **value** x, y;
real x, y, *re*, *ae*;
smaller : = $x < y \wedge$ *different* (x, y, *re*, *ae*);

Boolean procedure *greater* (x, y, *re*, *ae*); **value** x, y;
real x, y, *re*, *ae*;
greater : = $x > y \wedge$ *different* (x, y, *re*, *ae*).

Ces relations sont transitives :

smaller (x, y, *re*, *ae*) $\wedge$ *smaller* (y, z, *re*, *ae*) $\subset$ *smaller* (x, z, *re*, *ae*),
greater (x, y, *re*, *ae*) $\wedge$ *greater* (y, z, *re*, *ae*) $\subset$ *greater* (x, z, *re*, *ae*).

Un concept naturel est aussi que y est négligeable en comparaison de x, mais c'est déjà inclus. En effet y est négligeable en comparaison de x quand $x - y$ et $x + y$ sont égaux. Mais *equal* ($x - y$, $x + y$, *re*, *ae*) $\equiv$ *abs* (y) $\leqslant$ *delta* (x, *re*, *ae*), une relation qui montre clairement l'interprétation du concept *delta* (x, *re*, *ae*) comme maximum des quantités négligeables en comparaison de x.

Un aspect nouveau est introduit par la notion d'infinité et le concept de la limite. Fondamentale pour l'analyse numérique est la mesure d'incrédulité, c'est le nombre de fois *tim* qu'on doit constater en succession une propriété des éléments d'une suite infinie avant qu'on décide que les éléments suivants de la suite en jouissent de la même propriété.

Ce concept est nécessaire pour introduire le concept de la limite dans l'analyse numérique. Supposons donné une suite infinie des termes fk, numerotés par k, où fk est naturellement une expression en k. Nous disons que cette suite a une limite, quand on a constaté, commençant avec $k = a$, *tim* fois en succession l'égalité de deux termes en succession. Alors on a

real procedure *limit* (k, a, fk, *re*, *ae*, *tim*); **value** *tim*, *re*, *ae*;
integer k, a, *tim*; **real** fk, *re*, *ae*;
begin integer t; **real** f, g;
k : = a; f : = fk; t : = 0;
L : k : = $k + 1$; g : = fk;

```
    t : = if different (f, g, re, ae) then 0 else t + 1;
    if t = tim then limit : = g
        else begin f : = g; goto L end
    end limit.
```

On pourrait penser que cette définition de la limite est inacceptable parce que le critère que la différence entre deux termes successifs de la suite soit suffisamment petite ne garantit même pas l'existence d'une limite dans le sens classique, ni moins encore que'elle soit suffisamment approchée. Mais naturellement l'analyse numérique ne peut donner aucune garantie de cette sorte. Si le mathématicien veut avoir cette garantie, il se doit la procurer par le raisonnement mathématique. Du reste il peut obtenir avec la définition donnée chaque précision voulue en choisissant — dans le procedure statement — comme suite pas la suite à examiner mais une suite partielle formée par ces termes de la suite originale dont les indices accroîssent selon une loi ou l'autre plus rapidement que les nombres entiers, par exemple comme les multiples de dix, les nombres carrés, les puissances de deux ou bien les facultés.

Une application évidente est la somme d'une série convergente. La somme d'une série finie dont les termes fk dépendent sur k, lequel varie de a jusqu'à b est définie par

```
real procedure sum (k, a, b, fk); value a;
        integer k, a, b; real fk;
        if a > b then sum : = 0
            else begin k : = a; sum : = fk +
                                sum (k, a + 1, b, fk) end.
```

Alors la définition numérique de la somme *suminf* d'une série convergente pourrait être

```
real procedure suminf (k, a, fk, re, ae, tim);
        integer k, a, tim; real fk, re, ae;
        begin integer b;
        suminf: = limit (b, a, sum(k,a, b,fk), re, ae, tim) end.
```

Naturellement cette définition est assez crue parce que l'évaluation de la somme infinie exige de recalculer chaque fois la série finie commençant avec a au lieu d'additionner seulement un terme nouveau chaque fois.

Alors on a de nouveau un exemple, ici moins trivial, où une autre définition non récursive est avantageuse.

Une telle définition est

```
real procedure suminf (k, a, fk, re, ae, tim); value tim;
        integer k, a, tim; real fk, re, ae;
    begin integer t; real f, g;
        k := a; f := g := fk; t := 0;
        L: k := k + 1; g := g + fk;
        t := if different (f, g, re, ae) then 0 else t + 1;
        if t = tim then suminf := g
            else begin f := g; goto L end
        end suminf.
```

Intimement reliés avec la précision et l'incrédulité sont de divers problèmes concernants les fonctionnelles. Par example, quand on calcule avec une certaine précision donnée, une fonction $f(x)$ prend pour $a < x < b$ seulement un nombre fini des valeurs parce qu'il existe seulement un nombre fini de formes d'apparition de x dans un interval fini. Alors un problème familier comme la détermination avec une précision prescrite d'un zéro de $f(x)$, c.-à-d. une valeur de x, $a < x < b$ telle que $f(x) = 0$, est mal posé, même quand $f(a)$ et $f(b)$ ont des signes opposés, parce que l'existence d'une telle valeur de x est douteuse et en outre ne fait rien à l'affaire. Mais on peut définir ainsi : zéro de $f(x)$, donné que *sign* $(f(a)) \times$ *sign* $(f(b)) < 0$ est une valeur de x telle qu'on peut construire deux nombres x^1 et x^2 ainsi que

$a \leqslant x^1 \wedge x^1 \leqslant x \wedge x \leqslant x^2 \wedge x^2 \leqslant b \wedge$ *equal* $(x, x^1, re, ae) \wedge$ *equal* $(x, x^2, re, ae) \wedge$ *sign* $(f(x^1) \times$ *sign* $(f(x^2)) \leqslant 0$.

Cette définition d'un zéro est constructive parce qu'on peut construire des procédures qui définissent un tel nombre x. Pour finir nous donnons une telle procédure très efficace, qui est basée sur la méthode de regula falsi avec des raffinements. Cette méthode ne peut pas être décrite plus correctement que par le texte ALGOL 60 lui-même :

```
real procedure zero (x, a, b, fx, re, ae);
value a, b, re, ae; real x, a, b, fx, re, ae;
```

```
begin  real c, fa, fb, fc;
       switch dist := m2, m3, m1;
       x := a; fa := fx; x := b; fb := fx;
m1:    c := a; fc := fa;
m2:    if equal (b, c, re, ae) then
       begin zero := .5 × (b + c); goto m4 end;
       x := b + (b − a) × fb/(fa − fb);
       if sign (x − c) ≠ sign (b − x) then
       x := c + (c − b) × fc/(fb − fc);
       if equal (b, x, re, ae) then
       x := b + sign (c − b) × delta (.5 × (b + c), re, ae);
       a := b; fa := fb; b := x; fb := fx;
       goto dist [2 + sign (fc) × sign (fb)];
m3:    zero := b;
m4:
end zero.
```

LITERATURE

[1] J. W. BACKUS, F. L. BAUER, J. GREEN, C. KATZ, J. MCCARTHY, P. NAUR, A. J. PERLIS, H. RUTISHAUSER, K. SAMELSON, B. VAUQVOIS, J. H. WEGSTEIN; A. van WIJNGAARDEN, M. WOODGER : Report on the Algorithmic Language ALGOL 60 (editor P. Naur); *Numerische Mathematik* **2** (1960) pp. 106-137, Acta Polytechnica Scandinavica: *Math. and Comp. Mach. Ser.* no. 5 (1960); *Comm. Assoc. Comp. Mach.* **3** (1960), pp. 299-314.

[2] E. W. DIJKSTRA, Mathematisch Centrum, Amsterdam : Ein ALGOL-60-Uebersetzer für die *X*1; *Mathematik-Technik-Wirtschaft*, **8**. Jahrgang 1961, Heft 2 und 3.