A. van WIJNGAARDEN

NUMERICAL ANALYSIS AS AN INDEPENDENT SCIENCE

REPRINT

BIT

BIND 6, HEFTE NR. 1

KØBENHAVN 1966

BIT 6 (1966), 66–81

NUMERICAL ANALYSIS AS AN INDEPENDENT SCIENCE*†

A. VAN WIJNGAARDEN

Abstract.

The paper describes how a number of well-known mathematical concepts ought to be modified in order to make sense within the scope of numerical analysis. It is also shown how obvious difficulties can be overcome in a logical way. All algorithms suggested are given as ALGOL procedures.

1. Introduction.

The following is a sketch of numerical analysis considered as an independent science, i.e. independent of ordinary pure mathematics. Of course, the concepts with which numerical analysis deals, can all be considered as approximations to concepts in pure mathematics. A numerical analyst may, in this sense, compute an approximation to a zero of a function, to a limit, to an integral, to a derivative, and so on. However, he might also consider the result of the computation to be the thing that he wanted to have, and the "mathematical" concepts as approximations to his "numerical" concepts. This attitude might seem to be extremely narrow-minded for several reasons. First, there is little doubt that the framework of pure mathematics is considerably more impressive than that of numerical analysis. Moreover, a description of a numerical process alone does not define a result precisely. For this it is necessary also to define the computational tools precisely, i.e. the computer or arithmetic used, and it is not likely that anyone should like to base a science on a particular make of apparatus, presumably obsolete before long. On the other hand, however, with the rise of modern programming, algorithms of great generality and expressed in great rigour are becoming tools of the numerical analyst. In terms of these, he can express himself in a way which compares favourably with the way in which the pure mathematician expresses himself, and he might wish the concepts that he actually uses to form a consistent set themselves instead of considering them always as approximations to the classical concepts. That the actual execution of a computation described in terms of these algorithms will yield different results when different computers are used, is a fact which

* Communication MR 78 of the Computation Department of the Mathematical Centre, Amsterdam.

† This paper was presented at the NordSAM conference in Stockholm, Aug. 1964.

he has to live with anyhow. Actually, the margin of uncertainty in the result should explicitly play a role in the definition of the algorithms. Of course, in the definition of an algorithmic machine independent language, attention has to be paid to this question. E.g. the ALGOL 60 Report reads on this subject:

"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."

This, however, leaves open a number of questions. Actually, if no other information were given to the programmer, then he could not possibly write programs that make sense. In effect, an arithmetic in which an addition would yield the exact difference of the values of the operands, in which, moreover, a subtraction would yield the exact sum of those values, whereas the other operations would yield the exact proper results, satisfies the requirements, mentioned above, concerning "a finite deviation from the mathematically defined result". "The control of the possible consequences of such differences" seems out of the question under those circumstances. It was obviously the intention of the ALGOL authors that a little bit of good will was assumed in reading the sentence. The good will consists of the assumption that real arithmetic should not deviate too much from exact arithmetic. The point is, of couse, to define "too much". Such a definition should not contain arbitrary elements. In section 2 a sketch of a definition is given.

Having thus defined an arithmetic, the question arises to define basic concepts of comparison, given prescribed rates of tolerance. This is dealt with in section 3. With these concepts as a basis one can define concepts like sum, limit, infinite sum, zero of a function, integral, derivative, just like in ordinary analysis. This is dealt with in the following sections. It appears that a rather consistent approach to all these subjects can be made. Many of the concepts of exact analysis, like infinity and continuity have no proper counterpart in numerical analysis. On the other hand concepts like imprecision, incredulity, and efficiency, which are important parameters in numerical analysis have no proper counterpart in exact analysis.

2. Real arithmetic. Deviations in the small.

The arithmetic of real numbers will be defined by a number of axioms or rules. Any arithmetic that satisfies these rules is called a proper arithmetic. The arithmetic of integers and the evaluation of Boolean expressions other than relations is required to be exact.

In the following $e1$, $e2$, $e3$ are arithmetic expressions without side effects. More specifically, if in a formula the same expression occurs more than once, it is assumed that no values that enter into the evaluation of that expression have changed between the two evaluations. If they are specifically of type real, then they are denoted by $er1$, $er2$, $er3$, and if they are specifically of type integer then they are denoted by $ei1$, $ei2$, $ei3$. Variables are denoted by $v1$, $v2$.

The first rule

A1: $\vdash e1 = e1$

states that exact repetition of a computation yields the same result, i.e. that the finite deviation, mentioned above, does not depend on time. This is by no means obvious. It is a property of digital computation but not of analog computation.

The rules

A2: $\vdash +e1 = e1$,
A3: $\vdash (e1) = e1$

state that prefixing with a superfluous unary plus sign, where syntactically possible, and bracketing, do not change the value of an expression. These are, therefore, notations which do not correspond with arithmetical operations.

The seven rules

A4: $\vdash e1 = e2 \leftrightarrow e2 = e1$,
A5: $\vdash e1 \neq e2 \leftrightarrow \neg e1 = e2$,
A6: $\vdash e1 \neq e2 \leftrightarrow e1 > e2 \vee e1 < e2$,
A7: $\vdash \neg(e1 > e2 \wedge e1 < e2)$,
A8: $\vdash e1 > e2 \leftrightarrow e2 < e1$,
A9: $\vdash e1 \geqq e2 \leftrightarrow \neg e1 < e2$,
A10: $\vdash e1 \leqq e2 \leftrightarrow \neg e1 > e2$

define the properties of relations between two operands. They are in complete accordance with the usual definitions, but it is by no means obvious that they hold in every computing system. On the contrary, scrutiny may reveal exceptions here and there.

So far, nothing unusual is postulated. The first differences arise when transitivity is postulated. Strong transitivity is assumed to hold.

A11: $\vdash e1 > e2 \wedge e2 > e3 \rightarrow e1 > e3$.

Weaker rules, in which one or both of the inequality operators are replaced by the equality operator, however, need modification.

Indeed, the rules

$$\vdash e1 > e2 \wedge e2 = e3 \rightarrow e1 > e3\,,$$
$$\vdash e1 < e2 \wedge e2 = e3 \rightarrow e1 < e3$$

are too restrictive. For instance, if in a computer a real number (floating point number) and an integer (fixed point number) are both denoted by a machine word, then the precision of the real numbers is less than that of the integers in their common range, since both mantissa and exponent have to be stored in the word. Since, on the other hand, the range of the integers is less than that of the real numbers, it is likely that the comparison of a real number and an integer is performed by first converting the integer into a real number, since this transformation is always possible. This conversion then entails a loss of precision and both expressions $1.23456_{10}7 = 12345600$ and $1.23456_{10}7 = 12345601$ might have the value **true**. However, $12345601 > 12345600$ should also have the value **true**, since the integer arithmetic is exact. Hence,

$$12345601 > 12345600 \wedge 12345600 = 1.23456_{10}7 \leftrightarrow 12345601 > 1.23456_{10}7\,.$$

Therefore, the rules mentioned above are replaced by

A12: $\vdash e1 > er2 \wedge er2 = e3 \rightarrow e1 > e3$,
A13: $\vdash e1 < er2 \wedge er2 = e3 \rightarrow e1 < e3$,
A14: $\vdash e1 > ei2 \wedge ei2 = ei3 \rightarrow e1 > ei3$,
A15: $\vdash e1 < ei2 \wedge ei2 = ei3 \rightarrow e1 < ei3$.

Similarly the rule

$$\vdash e1 = e2 \wedge e2 = e3 \rightarrow e1 = e3$$

is too restrictive, since e.g.

$$12345600 = 1.23456_{10}7 \wedge 1.23456_{10}7 = 12345601 \leftrightarrow 12345600 = 12345601\,,$$

and is replaced by

A16: $\vdash e1 = er2 \wedge er2 = er3 \rightarrow e1 = er3$,
A17: $\vdash e1 = ei2 \wedge ei2 = ei3 \rightarrow e1 = ei3$.

The next operation to be investigated is the assignment. Let the expression $v1 := e1$ have the value **true** if the value most recently assigned to $v1$ is that of $e1$, and **false** otherwise. Then the rule

$$\vdash v1 := e1 \to v1 = e1$$

is too restrictive. For instance, in a computer in which real numbers are stored in a smaller precision than that of the floating arithmetic unit, whether simulated or actually present in hardware, assignment involves rounding or chopping. However, proper arithmetic requires that assignment be uniform and idempotent, i.e. that the following rules hold

A18: $\vdash v1 := e1 \wedge v2 := e1 \to v1 = v2$,
A19: $\vdash v1 := v2 \to v1 = v2$.

Next come the actual arithmetic operations, i.e. addition, subtraction, multiplication and division. Let $a1$ be one of the operators $+$, $-$, $\times$ or $/$. Then the substitution rule

$$\vdash e2 = e3 \to (e1)a1(e2) = (e1)a1(e3) \wedge (e2)a1(e1) = (e3)a1(e1)$$

is too restrictive, since e.g.

$$12345601 = 1.23456_{10}7 \mapsto$$
$$12345601 - 12345600 = 1.23456_{10}7 - 12345600 .$$

Proper rules are

A20: $\vdash ei2 = ei3 \to (e1)a1(ei2) = (e1)a1(ei3) \wedge (ei2)a1(e1) = (ei3)a1(e1)$,
A21: $\vdash er2 = er3 \to (e1)a1(er2) = (e1)a1(er3) \wedge (er2)a1(e1) = (er3)a1(e1)$,
A22: $\vdash ei2 = er3 \to (er1)a1(ei2) = (er1)a1(er3) \wedge (ei2)a1(er1) = (er3)a1(er1)$.

About the result of the arithmetical operations themselves the guiding principle is that the order relations which hold for the results of exact operations are weakened but not upset by replacing the exact operation by the corresponding numerical operations. For instance, it is a property of exact addition that

$$\vdash er1 > er2 \to er3 + er1 > er3 + er2 .$$

For numerical addition this is too restrictive, since e.g.

$$3.14_{10}50 + 2 = 3.14_{10}50 + 1$$

might very well hold true. However,

$$3.14_{10}50 + 2 < 3.14_{10}50 + 1$$

seems unacceptable.

Let a_e stand for the exact arithmetic operator, corresponding to the numerical arithmetic operator a. Let, moreover, r_w stand for the weak inequality operator $\geqq$ or $\leqq$ corresponding to the strong inequality operator r, i.e. $>$ or $<$ respectively. Then the following rules are postulated:

A23: $\vdash (e2r1\ 0 \to (e1)a1_e(e2)r2e1) \to (e1r1\ 0 \to (e1)a1(e2)r2_we1)$,
A24: $\vdash (e2r1\ 0 \to (e2)a1_e(e1)r2e1) \to (e1r1\ 0 \to (e2)a1(e1)r2_we1)$,
A25: $\vdash (e2r1e3 \to (e1)a1_e(e2)r2(e1)a2_e(e3)) \to (e2r1e3 \to (e1)a1(e2)r2_w(e1)a2(e3))$
A26: $\vdash (e2r1e3 \to (e2)a1_e(e1)r2(e3)a2_e(e1)) \to (e2r1e3 \to (e2)a1(e1)r2_w(e3)a2(e1))$

Since the left hand side does not explicitly contain a numerical arithmetic operator, it can be evaluated under certain circumstances using traditional means. If it turns out to have the value **true** then the right hand side has the value **true** which then yields a rule for proper arithmetic.

E.g. from A23 it follows that

$$\vdash (y > 0 \to x +_e y > x) \to (y > 0 \to x + y \geqq x)\,,$$

and since the left hand side has the value **true**

$$\vdash y > 0 \to x + y \geqq x\,.$$

Similarly, from A25 it follows that

$$\vdash (x > y \to z +_e x > z +_e y) \to (x > y \to z + x \geqq z + y)\,,$$

whence

$$\vdash x > y \to z + x \geqq z + y\,,$$

and similarly from A26 it follows that

$$\vdash x > y \to x + z \geqq y + z\,.$$

Again, one has from A24 e.g.

$$\vdash (x > 0 \to x \times_e 2 > x) \to (x > 0 \to x \times 2 \geqq x)\,,$$

whence

$$\vdash x > 0 \to x \times 2 \geqq x\,.$$

Real constants must, in this respect, be considered to stand for two integers separated by a division operator. For instance 3.1 stands for 31/10 or 310/100. If the division operation is considered to be exact, this is again denoted by the index e. For instance, from A25 one has

$$\vdash (314 > 310 \to 3.14_e > 3.1_e) \to (314 > 310 \to 3.14 \geqq 3.1)$$

and since $314 > 310$ has the value **true**

$$\vdash 3.14 \geqq 3.1 .$$

From A21 one has

$$\vdash 3.14 = 3.1 \to z + 3.14 = z + 3.1$$

and from $\vdash x > y \to z + x \geqq z + y$ one has

$$\vdash 3.14 > 3.1 \to z + 3.14 \geqq z + 3.1 .$$

Combining these three results, one has

$$z + 3.14 \geqq z + 3.1 .$$

However, about the three relations

$$3.1 < 3.10, \quad 3.1 = 3.10, \quad 3.1 > 3.10$$

the rules only state that one has the value **true** and the other two the value **false**.

At last, something must be postulated concerning the density of the real numbers in relation with the arithmetic operations. Of course, there may be an absolutely largest and absolutely smallest number in the set of real numbers. In fact, this is even more than likely the case. If a number is not itself that largest number, then it ought to be possible to get a larger number either by adding a sufficiently large number to it or by multiplying it with a sufficiently large number. Since the smallest number which is certainly not negligible with respect to a given number is that number itself, and since, moreover, doubling and halving a number are frequently used in numerical mathematics in order to enlarge or diminish a number, the following rules are postulated

A27: $\vdash e1 > 0 \to e1 + e1 > e1 \wedge e1 \times 2 > e1 \wedge 2 \times e1 > e1 \vee \forall e2, e2 \leqq e1$,
A28: $\vdash e1 < 0 \to e1 + e1 < e1 \wedge e1 \times 2 < e1 \wedge 2 \times e1 < e1 \vee \forall e2, e2 \geqq e1$,
A29: $\vdash e1 > 0 \to e1/2 < e1$,
A30: $\vdash e1 < 0 \to e1/2 > e1$.

A construction like e.g.

$$x := 3.14;\ l{:}x := 2 \times x;\ \textbf{if}\ x < xmax\ \textbf{then goto}\ l$$

will not give rise to infinite looping, whatever is the value of $xmax$, in virtue of these rules.

3. Tolerance and deviations in the large.

Having thus defined proper arithmetic, admitting explicitly small deviations between the results of arithmetical operations on real numbers executed with different arithmetics, attention is now paid to deviations in the large, deviations, that is to say, between the actual values obtained in the calculation and those that one should prefer to obtain, viz. the exact values. These deviations are partly caused by the accumulation of the effects of the small deviations in the arithmetical operations and partly by the approximations used, e.g. in replacing an infinite number of terms by a finite number of them. It is the task of the numerical analyst to control these deviations in the large.

This control is effected by putting a tolerance to the deviation between the ideal value of a quantity and the actual value obtained. This tolerance, on one hand, is dependent on the ideal value x, say, which is, of course, not under control, and on the other on the imprecision e, say, prescribed by the numerical analyst. This imprecision may be defined in several ways. It is assumed here that as actual value of the quantity whose ideal value is x, any number is admitted whose absolute deviation from x is at most the prescribed tolerance. This necessitates the rule

A31: $\vdash \mathit{tolerance}(x,e) \geqq 0$.

Moreover, it seems natural to require that if a number y is admitted as actual value of x, then y is admitted as actual value of any number of the interval $[x,y]$. In other words, both the upper bound and lower bound of admitted values should be monotonically not decreasing functions of the ideal value. Hence,

A32: $\vdash x1 < x2 \rightarrow x1 + \mathit{tolerance}(x1,e) \leqq x2 + \mathit{tolerance}(x2,e) \wedge$
$x1 - \mathit{tolerance}(x1,e) \leqq x2 - \mathit{tolerance}(x2,e)$.

Any definition of tolerance, which satisfies these two rules is acceptable for our purpose. As examples are mentioned:

i) e is the absolute error, $0 \leqq e$:

real procedure *tolerance*(x,e); **real** x,e; *tolerance* $:=$ e;

ii) e is the relative error, $0 \leqq e < 1$:

real procedure *tolerance*(x,e); **real** x,e; *tolerance* $:=$ *abs*$(x) \times e$;

iii) e is an array, whose first element is the relative error and whose second element is the absolute error with $0 \leqq e[1] < 1$ and $0 \leqq e[2]$:

```
real procedure tolerance(x,e); real x; real array e;
              tolerance := abs(x)×e[1]+e[2];
```

or

```
real procedure tolerance(x,e); real x; real array e;
              tolerance := max(abs(x)×e[1],e[2]);
```

where *abs* and *max* are defined by

```
real procedure abs(x); value x; real x;
              abs := if x ≧ 0 then x else −x;
real procedure max(x,y); value x, y; real x, y;
              max := if x ≧ y then x else y;
```

At first sight, this approach in which the tolerance is given as a function of the unknown ideal value rather than as a function of the known actual value may seem rather unpromising. However, the only question that makes sense in this respect is whether two actual values x and y are different within the imprecision e, i.e. whether there does not exist an ideal value z of which they both might be actual values. If $x=y$ the answer is obviously in the negative, since $z=x$ would do. Let otherwise $x<y$. Then, with $eps=(y-x)/2$, one has $x=(x+y)/2-eps$ and $y=(x+y)/2+eps$. If $eps \leqq tolerance((x+y)/2,e)$, then again the answer is shown to be in the negative, since $z=(x+y)/2$ would do. If, however, $eps > tolerance((x+y)/2,e)$ then there is no ideal value z which satisfies the conditions. Indeed, $z>(x+y)/2$ would imply $x<z-tolerance(z,e)$ in view of A32, and similarly $z<(x+y)/2$ would imply $y>z+tolerance(z,e)$. The question, whether x and y are different within the imprecision e is therefore answered by

```
Boolean procedure different(x,y,e); value x, y; real x, y;
                  different := abs((x−y)/2) > tolerance((x+y)/2,e);
```

It is seen that the unknown ideal value has completely disappeared from the result. The body of *different* contains some arithmetical operations and possibly also that of *tolerance*. This means that the prescribed imprecision must be large in comparison with the deviations due to the arithmetical operations, if, at least, the concept of two numbers being different shall have its intuitive meaning.

The converse question, i.e. whether x and y are equal within the imprecision e can only have the meaning whether it is impossible to prove that x and y must be different. Hence one has

```
Boolean procedure equal(x,y,e); equal := ¬different(x,y,e);
```

It should be stressed, that the arithmetical equality $x=y$ is of little

significance. Of course, $x=y \rightarrow equal(x,y,e)$, but $x \neq y$ does not tell anything about the equality of x and y within the imprecision e.

Similarly the questions whether x is smaller or greater than y within the imprecision e are answered by

Boolean procedure *smaller*(x,y,e); **value** x, y; **real** x, y;
smaller $:= x < y \wedge$ *different*(x,y,e);
Boolean procedure *greater*(x,y,e); **value** x, y; **real** x, y;
greater $:= x > y \wedge$ *different*(x,y,e);

Another useful concept is negligeability. A number x is said to be negligeable with respect to y within the imprecision e if adding x to y or subtracting x from y does not change its value, in other words, if $equal(y+x,y-x,e)$. Substitution in *equal* and *different* leads to

Boolean procedure *negligeable*(x,y,e);
negligeable $:=$ *abs*$(x) \leqq$ *tolerance*(y,e);

which definition sheds another light on the meaning of *tolerance*.

The last five procedures form a consistent set of comparisons that should take the place of the comparison operators in exact mathematics. The first four of them, *different*, *equal*, *smaller* and *greater* take the place of $\neq$, $=$, $<$ and $>$ respectively. The fifth one, *negligeable* has no counterpart.

4. Processes with memory.

In the preceding sections some consequences of the lack of definedness and precision of numerical analysis were discussed. However, there are also aspects of numerical analysis which do not have a counterpart in classical analysis, but are enrichments with respect to it. Among them is the fact that numerical analysis deals with processes rather than with static situations, and has correspondingly developed powerful means to describe those processes. By means of this description some familiar concepts can be given some more depth.

As an example the concept of a finite sum

$$\sum_{k=a}^{b} f_k$$

is considered. This corresponds presumably to the numerical concept defined by:

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

In some cases, however, this notation is too primitive, viz. if a number of values of the sum must be evaluated in which a stays constant, but b increases, a situation that often arises, as will be seen later on. The notation then does not show that a part of the sum has already been evaluated so that to the already known value of the sum some terms must be added. This can be remedied by letting the procedure dispose of some memory in which former results of activation can be remembered. Perhaps the most convenient way is to supply the procedure with two extra formal parameters d and ua. The actual parameter D corresponding to d is a real array with elements $D[1]$ and $D[2]$. In $D[1]$ the computed sum is remembered and in $D[2]$ the procedure remembers $b+1$. The actual parameter Ua corresponding to ua is of type Boolean and tells whether the actual parameter A corresponding to a has to be used. The procedure might be defined by

```
real procedure sumd(k,a,b,fk,d,ua); value a, b;
    begin sumd := d[1] := if ua then sum(k,a,b,fk)
                          else d[1]+sum(k,d[2],b,fk);
                  d[2] := b+1
    end;
```

The note-book D is, of course, interrogatable by the programmer, but if not interfered with, it is taken care of by the process itself. As an example,

$$\sum_{k=1}^{100}\left(\left\{\sum_{m=1}^{k}F(m)\right\}^2+\left\{\sum_{m=1}^{k}G(m)\right\}^2\right)^{\frac{1}{2}}$$

could be denoted by

$$sum\big(k,1,100,sqrt\big(sumd(m,1,k,F(m),D1,k=1)\uparrow 2 + sumd(m,1,k,G(m),D2,k=1)\uparrow 2\big)\big).$$

5. Limit and incredulity.

The next problem is the introduction of the concepts infinity and limit. Since there is, presumably, a largest number in the arithmetic, an obvious interpretation of infinity would be that largest number, and of the limit of a function, whose argument tends to infinity, the value of that function if its argument is that largest number. There are, however, several severe objections against this interpretation. First of all, it is not sure that there is a largest number in the arithmetic since all rules governing proper arithmetic are such that exact arithmetic satisfies them. Secondly, the argument which tends to infinity may very well be integer,

as, for instance, the index of a term in the case of an infinite series. Even if one would count this index with the help of a real variable, then adding one to it would presumably not lead to the largest number, but to a number which is not increased by adding one to it, which is much lower. At last, the determination of a limit would usually be hopelessly inefficient in this way. The solution is to make the concept limit also dependent on a prescribed imprecision e, viz. to define a limit as a number which is equal in the sense of section 3 to all terms with sufficiently high index:

$$(F = limit(k, fk, e)) \overset{\text{def}}{\Leftrightarrow} (\exists k_0, \forall k)(k > k_0 \rightarrow equal(fk, F, e)) .$$

This has still the drawback that a limit cannot be determined by computation only but necessitates mathematical reasoning, which in view of the vague definition of the arithmetic, is moreover much more difficult than in exact analysis. This last difficulty is overcome by the introduction of the concept *incredulity*. Incredulity is defined as the minimum number of times, *tim*, that a property of the elements of a sequence must have been observed in immediate succession in order to warrant the conclusion that all following elements have this property.

The incredulity employed in exact analysis is infinite in some sense. Human behaviour is founded, however, on a finite incredulity. At the very best the incredulity of a mathematician is somewhat greater than that of the non-mathematician since at the bottom of mathematical reasoning there is the belief that an argumentation which was valid until today will also be valid tomorrow. To determine the limit of a sequence of terms fk, i.e. a function of the index k, one has now only to compute for successive values of k to start with a suitably chosen value, the values of fk until a sequence of *tim* successive values of fk has been encountered each of which is equal (in the sense of section 3) to the last one. All following terms may, therefore, be supposed to have this property, so that the value of the last fk can be identified with the limit F in the definition given above: This is expressed by

```
real procedure limit(k, a, fk, e, tim); value tim; integer tim;
    begin integer i, j; real array f[1:tim];
        procedure next; begin f[i] := fk; k := k + 1 end;
        k := a;
        for i := 1 step 1 until tim do next;
        i := tim;
    L: for j := 1 step 1 until tim do
        if different(f[j], f[i], e) then
```

```
            begin i := i+1; if i > tim then i := 1;
                  next; goto L
            end;
            limit := f[i]
      end;
```

This definition of limit might, of course, be replaced by others, using some more sophistication. For instance, one might only require that a sequence of *tim* successive values of *fk* has been encountered each of which is equal to one of them or perhaps even equal to their mean. These requirements are easier to meet but more difficult to verify. If a limit is determined by using different methods, e.g. by using two different values of a in the definition given above, different results are to be expected. This does not create any contradiction. It is up to the numerical analyst to decide whether he wants to check and if so to decide what he wants to be done if different answers are obtained, in other words to decide whether he wants to include in a program something like

if *limit*$(n, 10, F(n), E, 5) \neq$ *limit*$(n, 100, F(n), E, 10)$ **then**

and if so what to write after **then**.

According to this definition of limit, it may even exist where it would not in exact analysis. For instance the value of *limit*$(n, 1, ln(n), E, 5)$ might be, depending on E, quite low. This again is nothing to be afraid of. On the contrary, a numerical calculation depending on the fact that the limit of $ln(n)$ does not exist, presumably would be rather doubtful. Moreover, the numerical analyst might, if it were important, use e.g. *limit*$(n, 1, ln(2 \uparrow n), E, 5)$, which would turn out to be much higher. An obvious application of the concept limit is the sum of an infinite series, which can be written down immediately as

```
real procedure suminf(k, a, fk, e, tim); value a; integer a;
    begin integer b; real array d[1 : 2];
          suminf := limit(b, a, sumd(k, a, b, fk, d, a=b), e, tim)
    end;
```

6. Function and continuity.

A real-valued function of one or more real-valued variables occurs in numerical mathematics as a recipe to compute a real value from the values of these variables. This recipe may be given as a simple expression or defined by a procedure which describes an arbitrarily intricate process

to be performed with and upon these variables. If, however, the arithmetic contains only a finite number of real numbers then even the most complicated procedure defines only a finite set of data and results. E.g. the function $x \times x - 2$ takes at most as many different values as x itself, and even if $1.4 \times 1.4 - 2 < 0$ and $1.5 \times 1.5 - 2 > 0$ both turn out to have the value **true**, then there is still no guarantee that the arithmetic contains a value x such that $x \times x - 2 = 0$: in an arithmetic working with 5 decimals and correct rounding one has

$$\begin{aligned} 1.41421 \times 1.41421 - 2.00000 &= -0.00001 < 0\,, \\ 1.41422 \times 1.41422 - 2.00000 &= 0.00002 > 0 \end{aligned}$$

and there is no number x such that $1.41421 < x \wedge x < 1.41422$.

This fact upsets thoroughly the concept of continuity and compels to scrutinize many definitions carefully.

As a fundamental example the equation $f(x) = 0$ may be taken. In exact analysis this defines a zero x of the function f, provided a value of x exists such that $f(x) = 0$. If, moreover, $f(x1) > 0$, $f(x2) < 0$ and $f(x)$ is continuous on the interval $[x1, x2]$, then there exists an $x \in [x1, x2]$ such that $f(x) = 0$. If $f(x)$ is not continuous on the interval, nothing can be said, e.g. if $f(x) =$ **if** $x \geqq 0$ **then** 1 **else** -1, then $f(1) > 0$, $f(-1) < 0$, but no x exists such that $f(x) = 0$.

In numerical analysis the situation is completely different. Of course, one cannot ask for a value of x such that $f(x) = 0$ but at the best for an approximation to it, i.e. a value y say, such that $equal(x, y, e)$. However, this equality can never be tested since x is unknown, and, even worse, may not exist! Since the only test on the quality of the result is to substitute it into the function and compute the value of the function, one might be tempted to accept y as result if $equal(f(y), 0, e)$. This, however, is unacceptable since $f(x) = 0$ defines x just as well as do $1000 \times f(x) = 0$ or $0.001 \times f(x) = 0$. A correct definition is

$$\begin{aligned} y = zero(x, fx, e) \overset{\text{def}}{\Leftrightarrow} (\exists\, a, b)(&equal(a, y, e) \wedge equal(b, y, e) \wedge \\ &x = a \to fx \geqq 0 \wedge x = b \to fx \leqq 0)\,. \end{aligned}$$

If two values of x, say $a1$ and $b1$ are known for which fx has not the same sign, then in several ways, e.g. by means of bisection or more efficient methods it is possible to construct numerically a sequence of intervals $[a1, b1] \supset [a2, b2] \supset \ldots \supset [a, b]$, and a value y which satisfies the relation mentioned above. A zero will therefore always be found under those conditions, even if in exact analysis this would not exist.

7. Integration and differentiation.

It now seems easy to define the value of an integral. Corresponding to

$$\int_a^b f(x)dx$$

one has according to the concept of a Riemann integral immediately:

```
real procedure integral(x, a, b, fx, e, tim); value a, b, real a, b;
    begin integer k, n;
        real procedure f(t); begin x := t; f := fx end;
        integral := limit(n, 1, sum(k, 1, n, f(a+(b-a)×k/n)), e, tim)
    end
```

There are, however, serious objections against this definition.

First of all, there is no guarantee that the argument

$$a+(b-a)\times k/n \in [a,b]$$

for all k, especially for $k=n$, since real arithmetic is employed. This might give rise to an undefined situation.

It can be remedied by having this argument computed by an auxiliary procedure declared inside the body of *integral*, which checks for this phenomenon. Much more serious is the inefficiency of the process. This might be held already against some former definitions, but here it is particularly striking. In pure mathematics efficiency is not an important point, but rigour and elegance are. In numerical analysis, however, efficiency is vital and at least as important as both rigour and elegance.

Since the terms of the sequence, whose limit tends to the integral, themselves can be considered as integrals of a stepwise constant approximating function, a definition in which the integral is represented as an infinite sum of corrections to a former result is more reasonable. Procedures which are both powerful and free from objections can and have been made.

Similarly, one might wish to define the value of the derivative of a function

$$\left(\frac{df(x)}{dx}\right)_{x=a}$$

e.g. as follows

```
real procedure derivative(x, a, fx, e, tim); value a; real a;
    begin integer n;
        real procedure f(t); begin x := t; f := fx end;
        derivative := limit(n, 1, (f(a+h/n)-f(a))/(h/n), e, tim)
    end;
```

Obviously, the same criticism of inefficiency applies. However, a more interesting deficiency of the definition given above concerns the control of the imprecision e. In the body a variable h occurs which governs the size of the variation in x around a. Obviously this h must have some value. However, it cannot be the task of the user of the procedure to assign a value to h, since the value he wants to compute has not to do with h at all. The procedure body, on the other hand, cannot assign a value to h independent of the actual parameters, since $h := 1$ might not work if a is so large that $a+h=a$ would hold, and the efficiency requires that h should be chosen as small as possible. Moreover, the subtraction $f(a+h/n)-f(a)$ entails a loss of precision with respect to the precision with which $f(a)$ is computed, a loss which increases when n increases. On the other hand in order to reach a higher precision in the result the value of n must increase. There is, therefore, a minimum set to the imprecision e that can be required. This difficulty provides, however, the clue how to choose the initial step h. Refined procedures concerned with this section will be published elsewhere.