Lecture 6 (hrs. 9,10) - October 2, 2025, 8:30 - 10:30 F3

(1.31) Definition (predefined functions).

Let M = F(β,m) be the set of machine numbers of the computer under consideration, and rd be the rounding function in M. The set FP of *predefined functions*, i.e. the functions that the computer can calculate by operating with the elements of M, is made up of three classes.

- The set of predefined functions corresponding to *arithmetic operations*. If $\cdot$ is one of the arithmetic operations between real numbers +, -, $\times$, / then the corresponding predefined function is indicated by the symbol $\odot$ (a small circle containing the symbol of the operation considered) and is defined, for each pair ξ, ϑ of elements of F(β,m) belonging to the domain of the operation $\cdot$, by

$$\xi \odot \vartheta = \mathrm{rd}(\xi \cdot \vartheta)$$

- The set of predefined functions corresponding to the usual *elementary functions* (sin, cos, arcsin, arccos, ln, exp ...). If f:A → R is one of the elementary functions then the corresponding predefined function is indicated by the symbol F and is defined, for each element ξ of F(β,m) belonging to the domain A of the elementary function f, by

$$F(\xi) = \mathrm{rd}(f(\xi))$$

- The set of predefined functions corresponding to *comparisons* between real numbers ($<$, $\leqslant$, $=$, $\neq$, $\geqslant$, $>$). In this case, since the elements of F(β,m) *are* real numbers, they are compared as such. Thus, the predefined functions corresponding to comparisons are simply the restrictions to F(β,m) $\times$ F(β,m) of comparisons between real numbers (and it is not necessary to introduce new symbols to indicate them).

(1.32) Definition (algorithm, naive algorithm).

Let $f_1,...,f_k$ be elementary functions or arithmetic operations and let f:A → R, where A is a suitable subset of R, be the function obtained by *composing* $f_1,...,f_k$:

$$f(x) = f_1 \circ \ldots \circ f_k(x)$$

(for example: f(x) = sin(x) + cos(x), where $f_3(x)$ = sin(x), $f_2(x)$ = cos(x) and $f_1(x_1,x_2) = x_1 + x_2$). If we ask *Scilab* to evaluate the function f with the instruction

```
> f(x)
```

the returned value will be

$$F_1 \circ \ldots \circ F_k(\mathrm{rd}(x))$$

where $F_1,...,F_k(x)$ are, respectively, the predefined functions corresponding to $f_1,...,f_k(x)$.

The expression $F_1 \circ \ldots \circ F_k(\mathrm{rd}(x))$ defines a function φ: A → M called the *naive algorithm* for f (for the function in the example: $\varphi(x) = \mathrm{SEN}(\mathrm{rd}(x)) \oplus \mathrm{COS}(\mathrm{rd}(x))$, defined for every x in R). The term *algorithm* generally refers to a *finite* sequence of operations for calculating predefined functions.

Except that very special cases, there will be values of x for which $f(x) \neq \varphi(x)$. In these cases, we use $\varphi(x)$ to approximate f(x), and it is interesting to have *information on the error committed.*

To obtain this information we introduce the notions of *accurate algorithm*, *stable algorithm* and *well-conditioned computation of the value of a function.*

(1.33) Definition (accurate algorithm).

Let $f:A \to R$ be a cunction, $\varphi:A \to M$ the algorithm used to approximate the values of f and $x \in A$.

The algorithm φ is said to be *accurate* (when used to approximate the value of f at x) if there exists a real number ε such that:

(1) $\varphi(x) = (1 + \varepsilon) f(x)$
(2) ε 'small'

If the algorithm is accurate for every $x \in B \subset A$, the algorithm is said to be accurate on B. In that case ε will depend on x.

(1.34) Remark.

- Let f and x be such that $f(x) \neq 0$. Condition (1) of the previous Definition is *equivalent* to the following:

$$\varepsilon = \frac{\varphi(x) - f(x)}{f(x)}$$

  In this case, then, the algorithm is accurate is equivalent to saying that the relative error committed by approximating f(x) with $\varphi(x)$ is 'small'.

- If the algorithm is accurate we have: $f(x) = 0 \Leftrightarrow \varphi(x) = 0$.

- The definition of accurate algorithm is *qualitative* because the term 'small' relative to ε is not quantified. The concrete meaning of the term 'small' depends on the individual case. For example, if, as in the case of the bisection method, it only matters that $\varphi(x)$ and f(x) have the same sign, ε 'small' means $\varepsilon > -1$.

  Exercise: We use λ to approximate $L > 0$. What relative error ε is made using $\lambda = 0$? What value of λ should be used to obtain a relative error $\varepsilon = 1$?

(1.35) Definition (stable algorithm).

Let $f:A \to R$ be a function, $\varphi:A \to M$ be the algorithm used to approximate the values of f and $x \in A$.

The algorithm φ is said to be *stable* (when used to approximate the value of f at x) if there exist real numbers ε_a, ε_v such that:

(1) $\varphi(x) = (1 + \varepsilon_v)\, f((1 + \varepsilon_a)x)$

(2) ε_a, ε_v 'small'

If the algorithm is stable for every $x \in B \subset A$, we say that the algorithm is stable on B. In this case ε_a, ε_v will depend on x.

(1.36) Remark.

- If an algorithm is accurate then it is stable ($\varepsilon_a = 0$, $\varepsilon_v = \varepsilon$); a stable algorithm *may not* be accurate.
- Informally: a stable algorithm returns a *good approximation* (ε_v 'small') of the value of f at a point *close* to x (ε_a 'small').

(1.37) Remark ('good' algorithm).

The notion of stability formalizes the idea of a '*good*' algorithm for approximating the values of a given f. For example, if f is an elementary function and φ is the naive algorithm for f, then, calling F the predefined function corresponding to f, we have:

$$\varphi(x) = F(\, rd(x)\,) = rd(\, f(\, rd(x)\,)\,)$$

(1.38) Theorem (relative error and perturbation).

Recalling the Definition of relative error committed by approximating a real number t with its rounded rd(t) and Theorem (1.28) of Lesson 5 on the limitation of the relative error, we obtain:

Let x be a real number and rd be the rounding function in $F(\beta,m)$. There exists a real number ε such that:

$$rd(x) = (1 + \varepsilon)x \quad \text{and} \quad |\varepsilon| < u$$

The equality expresses the rounded of x as a (small) *multiplicative perturbation* of x.

(Proof: if $x \neq 0$ then ε is the relative error committed by approximating x with rd(x); if x = 0 (and therefore rd(x) = 0) the equality holds, for example, with $\varepsilon = 0$.)

(1.39) Remark (continuation of the previous one).

Using the previous Theorem twice we finally get:

$$\varphi(x) = (1 + \varepsilon_2) f(\, (1 + \varepsilon_1)x\,) \quad \text{with} \quad |\varepsilon_1| < u \ \text{ e } \ |\varepsilon_2| < u$$

The algorithm φ returns *the best possible approximation* of the value of f at the point *closest to* x. In this sense, φ is the 'best possible' algorithm that the computer can use to approximate f(x). Hence, generalizing, the idea that a 'good' algorithm for approximating the value of a function at a given point is an algorithm that returns a good approximation of the value of the function at a point close to the one where we wanted to compute it.

(1.40) <u>Definition</u> (well-conditioned computation of the value of a function).

Let $f:A \to R$ be a function and $x \in A$. The computation of the value of f at x is *well-conditioned* <u>if</u>: for *every* 'small' real number α *there exists* a 'small' real number ε_V such that

$$f(\ (1 + \alpha)x\ ) = (1 + \varepsilon_V)f(x)$$

Informally: the computation of the value of f at x is well conditioned if the value of f at any point 'near' x is a 'good' approximation of the value of f at x.

(1.41) <u>Remark</u>.

- The property that the computation of the value of f at x is well-conditioned concerns *only* the function f. In particular, it is not related to the *algorithm* chosen to approximate the values of f.
- If $f(x) \neq 0$, the value of ε_V, once α is assigned, is *determined*. Specifically, ε_V is:

$$\varepsilon_V = \frac{f(\ (1 + \alpha)x\ ) - f(x)}{f(x)}$$

(1.42) <u>Theorem</u> (stability + well-conditioning => accuracy).

Let $f:A \to R$ be a function, $x \in A$, and φ be the algorithm used to approximate f(x). <u>If</u> the algorithm is *stable* and the computation of f at x is *well-conditioned*, <u>then</u> the algorithm is *accurate*.

<u>Proof</u>. By the stability of the algorithm there exist ε_1 and ε_2 such that:

$$\varphi(x) = (1 + \varepsilon_2)f(\ (1 + \varepsilon_1)x\ ) \quad \text{and} \quad \varepsilon_1 \text{ e } \varepsilon_2 \text{ 'small'}$$

By the well-conditioning of the calculation of f at x there exists ε_3 such that:

$$f(\ (1 + \varepsilon_1)x\ ) = (1 + \varepsilon_3)f(x) \quad \text{and} \quad \varepsilon_3 \text{ 'small'}$$

Then we can rewrite:

$$\varphi(x) = (1 + \varepsilon_2)(1 + \varepsilon_3)f(x)$$

and, introducing $(1 + \varepsilon_2)(1 + \varepsilon_3) = 1 + t$, i.e. $t = \varepsilon_2 + \varepsilon_3 + \varepsilon_2\varepsilon_3$, we get:

$$\varphi(x) = (1 + t)f(x) \quad \text{and} \quad t \text{ 'small'}$$

so the algorithm is accurate.

(1.43) Remark (stability of naive algorithms in elementary cases).

- From what we deduced in Remarks (1.37) and (1.39), if $f:A \to R$ is an elementary function and φ is the naive algorithm for f, φ is stable on A: *for each elementary function the naive algorithm is stable.*
- Let $f(x_1,x_2) = x_1 + x_2$. The naive algorithm for f is:

$$\varphi(x_1,x_2) = rd(x_1) \oplus rd(x_2)$$

Recalling the definition of $\oplus$ (see Definition (1.31)) and using Theorem (1.38) three times we obtain:

$$\varphi(x_1,x_2) = (1 + \varepsilon_3)(\ (1 + \varepsilon_1)x + (1 + \varepsilon_2)x\ ) \quad , \quad \text{with } |\varepsilon_j| \leqslant u\ ,\ j = 1,2,3$$

So, the naive algorithm for the sum is stable.

Similarly, *the naive algorithm for each of the arithmetic operations is shown to be stable.*