Computable Analysis

Merging fundamental concepts of analysis and recursion theory, we may ask:

(1) Is the exponential function computable?

(2) Are union and intersection of closed subsets of the real plane computable?

(3) Are differentiation and integration computable operators?

(4) Is zero-finding for complex polynomials computable.

For a deep understanding and for future development of computation in analysis, a sound theoretical foundation is indispensible. Computable Analysis is developed as the theory of those functions on the real numbers (and other sets from analysis) which can be computed by machines.

We merge the concepts of limit and approximation with machine models and discrete computation.

Type-1 Theory of Effectivity

While analysis and numerical analysis have a very long tradition (Gauss and Lagrange were experts in numerical computation), it was not until the 1930s that S. Kleene, A. Church, A. Turing and others proposed various definitions of "effectively calculable" functions on the natural numbers. There is now a well-established and rich theory of computability and complexity for functions on the natural numbers or on finite words.

Although a number of authors also studied computability on the real numbers, computable analysis is still underdeveloped. Moreover, there are several non-equivalent suggestions of how to model effectivity in analysis, and in particular, computability of real functions.

~~K.-I. Goto was the first author to introduce computable real numbers. Since that time,~~

Type-2 Theory of Effectivity

A. Turing (1936) was the first to introduce computable real numbers. Since that time, computable analysis has been developed continuously. Among the large number of publications, there are some books on computable analysis (or related topics) such as R.L. Goodstein (1959), D. Klaua (1961), S. Mazur (1963), O. Aberth (1980), B. Kushner (1984), E. Bishop and D. Bridges (1985), K. Weihrauch (1987), M. Pour-El and J. Richards (1989), J. Traub, et al (1988), K. Ko (1991), and L. Blum et al (1998).

All these books have important concepts in common, but differ in their contents and technical framework, and each author presents the topic from their individual point of view. This mirrors the fact that computable analysis has no generally accepted foundation.

K. Weihrauch (2000) presents a new coherent foundation of computable analysis, rooted in the definition of computable real functions based on the work by A. Grzegorczyk (1955) and J. Hauck (1973, 1978, 1980, 1981, 1982). To distinguish it from others, he called it "Type-2 Theory of Effectivity" (TTE).

(4.4)

Notational Conventions

We will include 0 in the natural numbers, denote by 2^A all subsets of A, and by $E(A)$, all finite subsets of A. By convention, a product of 0 sets will be $\{()\}$ where $()$ is the empty tuple.

A "correspondence" or "multivalued partial function" from A to B is a triple:

$$f = (A, B, R_f)$$

such that $R_f \subseteq A \times B$, where A is called the source, B the target, and R_f the graph of f. For $X \subseteq A$, we define the image of X under f by:

$$f[X] := \{b \in B \mid \exists a \in X,\ (a,b) \in R_f\}.$$

We define the inverse of f:

$$f^{-1} := (B, A, R_f^{-1})$$

where $R_f^{-1} := \{(b,a) \mid (a,b) \in R_f\}$.

We define the range and domain of f:

$$\text{range}(f) := f[A]$$

$$\text{dom}(f) := f^{-1}[B].$$

Notation Continued

We will denote a "correspondence" f from A to B by $f :\subseteq A \rightrightarrows B$.

A "partial function" $f :\subseteq A \to B$ from A to B is a multi-valued function $f :\subseteq A \rightrightarrows B$ such that $f[\{a\}]$ contains exactly one element for each $a \in \mathrm{dom}(f)$.

A total function $f : A \to B$ from A to B is a partial function $f :\subseteq A \to B$ such that $\mathrm{dom}(f) = A$. The set of all total functions $f : A \to B$ is denoted by B^A.

For a partial function $f :\subseteq A \to B$, $f(a)$ denotes the single element from $f[\{a\}]$ if $a \in \mathrm{dom}(f)$. Otherwise, $f(a) = \perp$.

For multi-valued functions $f_i :\subseteq A \rightrightarrows B_i$, define $(f_1, \ldots, f_k) :\subseteq A \rightrightarrows B_1 \times \cdots \times B_k$ by:

$$(f_1, \ldots, f_k)[\{a\}] := f_1[\{a\}] \times \cdots \times f_k[\{a\}].$$

For multi-valued functions $f :\subseteq A \rightrightarrows B$ and $g :\subseteq B \rightrightarrows C$, define the composition $g \circ f :\subseteq A \rightrightarrows C$:

$$a \in \mathrm{dom}(g \circ f) \iff a \in \mathrm{dom}(f) \wedge f[\{a\}] \subseteq \mathrm{dom}(g)$$
$$(g \circ f)[\{a\}] := g[f[\{a\}]] \qquad \forall a \in \mathrm{dom}(g \circ f).$$

$\mathrm{id}_X : X \to X$ denotes the identity function on X.

(1.6)

Words and Concatenation

For any set Σ, Σ^n denotes the set of all words over Σ of length n, $\Sigma^{\leq n}$ the set $\Sigma^0 \cup \cdots \cup \Sigma^n$, and Σ^* the set of all finite words over Σ. We denote by $|w|$ the length of word w, and by λ the word of length 0. By Σ^ω we denote the set $\{p \mid p : \mathbb{N} \to \Sigma\}$ all infinite sequences over Σ.

Concatenation in $\Sigma^* \times \Sigma^*$ is defined in the obvious way, and can be extended to $\Sigma^* \times \Sigma^\omega$. If $x = uvw \in \Sigma^*$ and $q = uvp \in \Sigma^\omega$, then u is a prefix of x and q ($u \sqsubset x$, $u \sqsubset q$). V is a subword of x and q ($V \vartriangleleft x$, $V \vartriangleleft q$). A is called prefix-free iff $\forall x, y \in A$, $x \not\sqsubset y$. By $p_{\leq n}$ we denote the prefix of length n of p. If $A, B \subseteq \Sigma^*$ (or $B \subseteq \Sigma^\omega$), then we define:

$$AB = \{uv \mid u \in A,\ v \in B\}.$$

We may also write $A^n = AA \cdots A$. It will be clear from the context if this means Cartesian or concatenation product.

When Σ is a non-empty finite set, we may call it an alphabet, and write words over that alphabet in double quotes.

Numberings

Ordinary (Type-1) recursion theory considers the computable number functions $f :\subseteq \mathbb{N}^k \to \mathbb{N}$ and the computable word functions $g :\subseteq (\Sigma_1^*)^k \to \Sigma_1^*$. Computability can be transferred to other sets M by means of "numberings". A numbering of a set M is a surjective function $\nu :\subseteq \mathbb{N} \to M$.

For a given alphabet $\Sigma_1 = \{a_1, \ldots, a_n\}$, define the bijective standard numbering $\nu_{\Sigma_1} : \mathbb{N} \to \Sigma_1^*$ of Σ_1^* as follows:

$$\nu_{\Sigma_1}^{-1}(\lambda) := 0$$
$$\nu_{\Sigma_1}^{-1}(a_{i_k} \cdots a_{i_0}) := i_k \cdot n^k + \cdots + i_0 \cdot n^0.$$

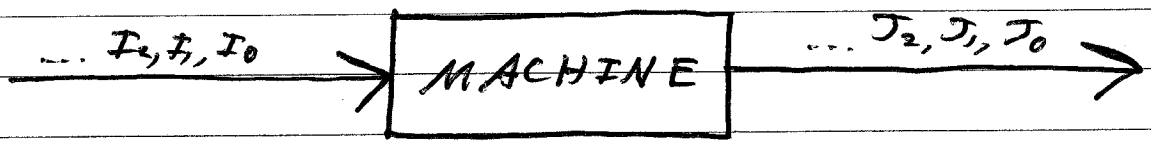
Then, $f :\subseteq \mathbb{N} \to \mathbb{N}$ is computable iff $\nu_{\Sigma_1} \circ f \circ \nu_{\Sigma_1}^{-1} :\subseteq \Sigma_1^* \to \Sigma_1^*$ is computable (and correspondingly for $f :\subseteq \mathbb{N}^m \to \mathbb{N}$).

The bijective Cantor pairing function $\langle , \rangle : \mathbb{N}^2 \to \mathbb{N}$ defined by $\langle x, y \rangle := y + (x+y)(x+y+1)/2$ is computable, as well as the projections $\langle \rangle_1, \langle \rangle_2$ of its inverse. For $n > 2$ arguments, we define it inductively. This will allow us to extend the definition of computable functions to functions of mixed type by calling $\nu_{\Sigma_1}, \nu_{\Sigma_1}^{-1}$ computable and closing under composition (e.g. $A \subseteq \Sigma_1^* \times \mathbb{N} \times \Sigma_1^*$).

A Sketch of TTE

Type-2 Theory of Effectivity extends ordinary (Type-1) computability and complexity theory. TTE admits two levels of effectivity: continuity, and computability as a specialization of continuity. We will present an overview of TTE informally, without using a mathematically precise model of computation.

Clearly Type-1 namings (names) are not sufficient to name the reals, as $\mathbb{N}$ is only countable. However, real numbers can be represented by infinite sequences (for example, by infinite decimal fractions). In TTE, infinite sequences are used as names of real numbers, and machines which transfer infinite sequences to infinite sequences are used to compute (real) numbers.



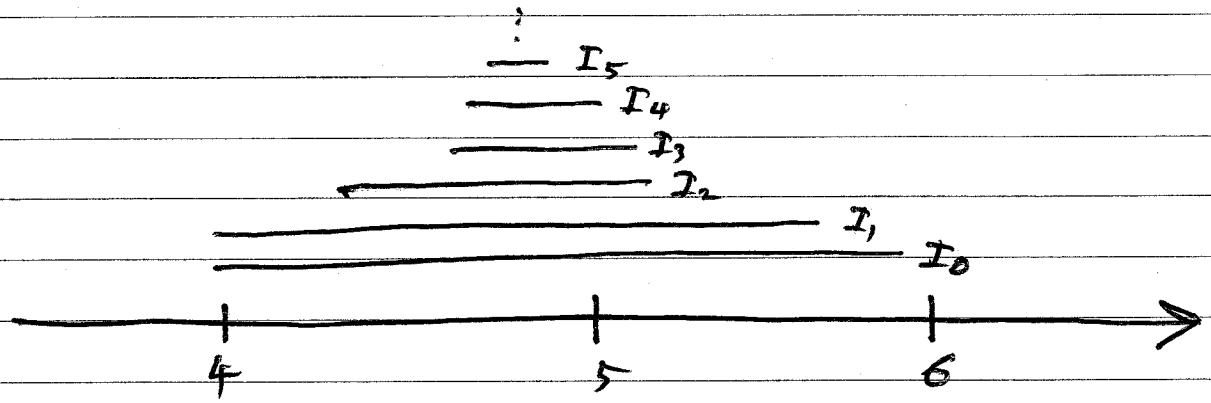
On input $(I_0, I_1, I_2, \ldots)$, from time to time, the machine reads a new sequence element I_k from its input stream, and from time to time, it writes a new sequence element J_m to its output stream. I_{k+1} must be read after I_k, and J_{m+1} must be written after J_m.

A Naming System for $\mathbb{R}$

It turns out that infinite decimal fractions induce a computability concept that is not very interesting. Instead, we can use infinite sequences of nested intervals as names.

For now, we define a name of a real number $x \in \mathbb{R}$ to be a sequence $(I_0, I_1, \ldots)$ of closed rational intervals $[a, b]$ $(a < b, a, b \in \mathbb{Q})$ such that $I_{n+1} \subseteq I_n$ for all $n \in \mathbb{N}$ and:

$$\bigcap_{n \in \mathbb{N}} I_n = \{x\}.$$



We assume, tacitly, that intervals are encoded appropriately, such that, strictly speaking, a name of a real number is an infinite sequence of symbols.

(1.10)

Computable Real Numbers

We call a real number computable iff it has a computable name.

Eg 1 Every rational number $r \in \mathbb{Q}$ is computable.

Proof: Define $I_n = [r - 2^{-n}, r + 2^{-n}]$ for all $n \in \mathbb{N}$. Then the sequence:

$$(I_0, I_1, I_2, \ldots)$$

is a computable name of r. □

Eg 2 $\sqrt{2}$ is computable.

Proof: Define $f : \mathbb{N} \to \mathbb{N}$ by $f(n) := \min\{k \in \mathbb{N} \mid k^2 \leq 2n^2 \leq (k+1)^2\}$, $J_0 := [1, 2]$, $J_n := [f(n)/n, (f(n)+2)/n]$, $n > 0$.

Then, the sequence $(J_0, J_1, \ldots)$ is computable, $\sqrt{2} \in J_n$ for all n, and $\lim_{n \to \infty} \text{length}(J_n) = 0$. However, the sequence is not nested in general. We can fix this by defining $I_n := J_0 \cap J_1 \cap \cdots \cap J_n$. Then $(I_0, I_1, \ldots)$ is a computable name of $\sqrt{2}$. □

Eg 3 $\log_3(5)$ is computable.

Proof: $f(n) := \min\{k \in \mathbb{N} \mid 3^k < 5^n < 3^{k+1}\} \ldots$

Computable Real Functions

We call a real function $f :\subseteq \mathbb{R} \to \mathbb{R}$ computable iff some machine maps any name of $x \in dom(f)$ to a name of $f(x)$. For real functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}$, we consider machines reading n names in parallel.

Notice that the machine must behave correctly only for every name of $x \in dom(f)$. For other sequences, it may behave arbitrarily.

It turns out that the elementary real functions exp, sin, log, arcsin ... are computable, and that computable real functions map computable numbers to computable numbers. Moreover, computable real functions are closed under composition.

Eg 3. Real multiplication $(x, y) \mapsto x \cdot y$ is computable.

Proof: For closed $\mathbb{Q}$-intervals I, J, define $I \cdot J = \{x \cdot y \mid x \in I, y \in J\}$. There is a machine with two input streams which maps inputs $(I_0, I_1, \ldots), (J_0, J_1, \ldots)$ to $(I_0 \cdot I_0, I_1 \cdot J_1, \ldots)$. If the first is a name of x, the second of y, then clearly the output is a name of $x \cdot y$. □

Example 4

The square root $\sqrt{\cdot} :\subseteq \mathbb{IR} \to \mathbb{IR}$ is computable.

Proof: Since, for example, $\sqrt{[2,3]}$ is not a rational interval, we must modify the proof of Example 3 for it to work here.

We map each input interval I to a rational interval which is slightly longer than $\sqrt{I}$. For any $a, b \in \mathbb{Q}$ s.t. $0 \leq a < b$, there are rational numbers $r, s \geq 0$ s.t.:

$$a - (b-a) < r^2 \leq a < b \leq s^2 < b + (b-a).$$

Therefore there is a computable function f which for each rational interval I, with no negative elements, determines a rational interval $K = f(I)$ such that $I \subseteq K^2$ and $\text{length}(K^2) < 3 \cdot \text{length}(I)$.

For any rational interval $[a,b]$ with $b \geq 0$, let $g[a,b] := [\max(0,a), b]$. If $(I_0, I_1, \ldots)$ is a name of $x \geq 0$, then $((f \circ g)[I_0], (f \circ g)[I_1], \ldots)$ is a sequence of rational intervals that converges to $\sqrt{x}$. The sequence is not necessarily nested, but we can fix this just as in Example 3. $\square$

Theorem (Continuity)

Every computable real function is continuous!

Proof sketch: We look only at the case $f:\mathbb{R}\to\mathbb{R}$.

Let M be a machine computing f, and let $x\in\mathbb{R}$. We will show f is continuous at x, and thus is continuous, since the choice of x was arbitrary. That is, we will show that $\forall V\in T_{\mathbb{R}},\ f(x)\in V \Rightarrow \exists U\in T_{\mathbb{R}},\ x\in U\wedge f[U]\subseteq V$.

So, let V be an open set with $f(x)\in V$. The real number x has a name $([a_0,b_0],[a_1,b_1],\dots)$ such that $a_{i+1}<a_i<b_{i+1}<b_i$. Let $(J_0,J_1,\dots)$ be the name that M produces on this input. Then $J_n\subseteq V$ for some $n\in\mathbb{N}$. Suppose M took k steps to produce the output sequence up to J_n. Then, M can certainly only have read at most k input intervals.

Simply choose $U:=(a_k,b_k)$. Clearly $x\in U$. Suppose $y\in U$. Then y has a name of the form $([a_0,b_0],[a_1,b_1],\dots,[a_k,b_k],L_{k+1},L_{k+2},\dots)$. On this input, the machine M writes also $(J_0,\dots,J_n)$ within the first k steps, which is the initial part of the name of $f(y)$. Thus $f(y)\in J_n\subseteq V$. Thus $f[U]\subseteq J_n\subseteq V$ as required.

So f is continuous. □

Important Remarks

In our proof of the last theorem, we have used the essential observation that any finite portion of the output of a computation is determined already by a finite portion of its input. However, we did not use that the transformation from inputs to outputs is computable.

As a consequence of this theorem, simple real functions like the step function:

$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

and the Gauss staircase $g(x) = \lfloor x \rfloor$ are not computable! These functions are easily definable in our mathematical language, but this does not necessarily imply computability.

As far as we know, the step function, or indeed any non-continuous function, cannot be computed by physical devices.

N.B. One of the reasons there is no agreement on the foundations of computable analysis is the issue that this definition does not say that such simple functions are computable.

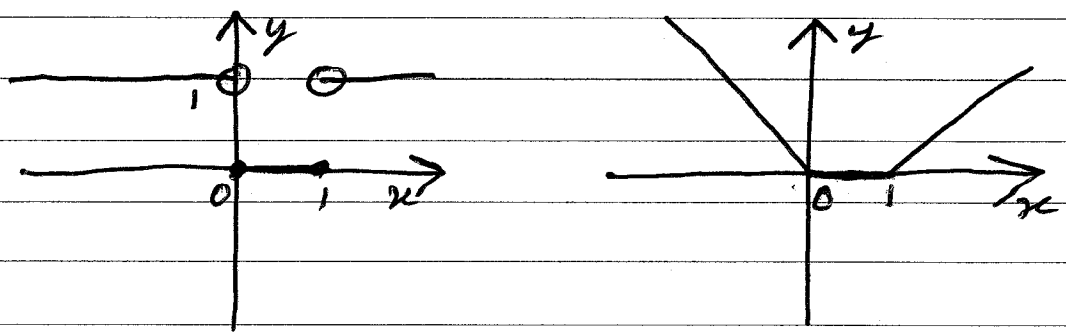
Subsets of $\mathbb{R}$: Recursive

A subset $A \subseteq \mathbb{N}$ is recursive (decidable) iff its characteristic function $cf_A : \mathbb{N} \to \mathbb{N}$ is computable. Suppose we extended this definition to $\mathbb{R}$. Then, by the continuity theorem, the only computable subsets would be $\emptyset$ and $\mathbb{R}$, so that definition is useless.

The distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ defined by $d_A(x) := \inf_{y \in A} |y - x|$ is a more useful generalization of the discrete characterisation function. We call a (closed) subset $A \subseteq \mathbb{R}^n$ recursive iff its distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ is computable.

It turns out that closed intervals $[a,b]$ with computable endpoints, and the graph of any computable $f : \mathbb{R} \to \mathbb{R}$ are recursive.

Eg 5 Compare the graphs of $1 - cf_{[0,1]}(x)$ and $d_{[0,1]}(x)$:



Subsets of $\mathbb{R}$: Computability

For defining computability on subsets of $\mathbb{R}$, we introduce naming systems of sets of subsets. Unfortunately $2^{\mathbb{R}}$ is too large, so we must restrict ourselves. For now, we will only consider the open subsets of $\mathbb{R}$: $\mathcal{O}(\mathbb{R})$ $(= \tau_{\mathbb{R}})$.

We define a name of an open subset $U \subseteq \mathbb{R}$ to be a sequence $(I_0, I_1, \ldots)$ of open intervals with rational boundaries such that $U = I_0 \cup I_1 \cup I_2 \cup \cdots$ In the general case, $U \subseteq \mathbb{R}^n$, each I_k is a product of n open intervals with rational boundaries. We call an open set recursively enumerable iff it has a computable name. As for real functions, we call a function on $\mathcal{O}(\mathbb{R})$ computable iff some machine maps names of arguments to names of results.

Eg 6 The open intervals (a, b) with computable endpoints are recursively enumerable. So is the set $\{(x, y) \subseteq \mathbb{R}^2 \mid x < y\}$. In fact $\{(x, y) \subseteq \mathbb{R}^2 \mid x \leq y\}$ is recursive.

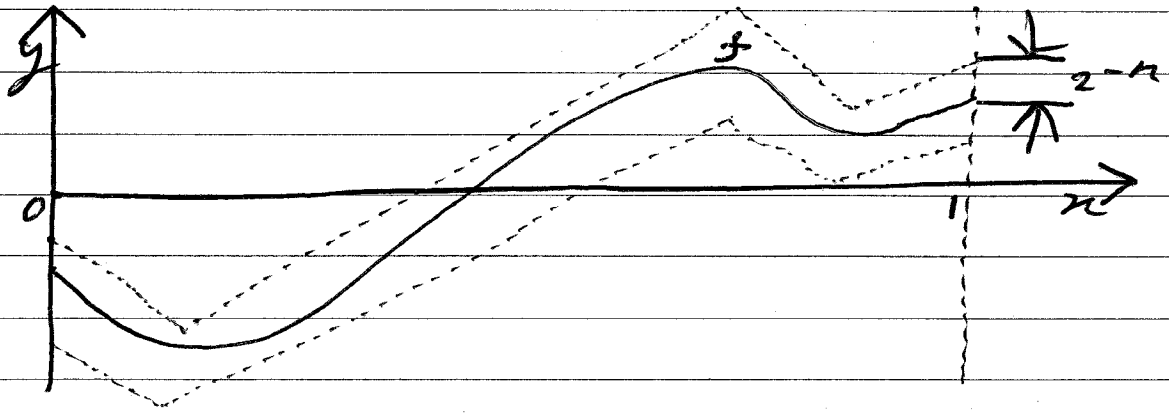
Eg 7 The functions intersection and union on open sets are computable.

The Space $C[0,1]$

We call $f:[0,1]\to\mathbb{R}$ a rational polygon iff its graph is a polygon with finitely many vertices, with $\mathbb{Q}$-coordinates. Define the function ball with center $p\in C[0,1]$, radius $r>0$: $B(p,r) := \{f\in C[0,1] \mid d(f,p)<r\}$ where $d(f,p)=\max_{x\in[0,1]}|f(x)-p(x)|$.

A name of a continuous function $f:[0,1]\to\mathbb{R}$ is a sequence $(B_0, B_1, \dots)$ where B_n is a function ball of radius 2^{-n} with $f\in B_n$ the center of which is a rational polygon.

Eg 8



A name $(B_0, B_1, \dots)$ of f encloses f arbitrarily narrowly. A function $f:[0,1]\to\mathbb{R}$ can have a computable name, and it can be computable (by a machine transforming each name of the real numbers $x\in[0,1]$ to a name of $f(x)$). It will turn out these notions are equivalent!

Closing Remarks

Our naming systems of $\mathbb{R}$ and $C[0,1]$ make the evaluation function:

$$Apply: C[0,1] \times [0,1] \to \mathbb{R}$$

defined by $Apply(f, x) := f(x)$, computable. That is, there is a machine which transforms any name of any f and any name of any x to a name of $f(x)$.

There are many other naming systems of $C[0,1]$ for which the evaluation function becomes computable, however among all of these, this is the "weakest".

Eg 9 Integration $f \mapsto \int_0^1 f(x)\,dx$ is an important computable operator (when $f \in C[0,1]$), while differentiation $f \mapsto f'$ for continuously differentiable $f \in C[0,1]$ is not computable.

Eg 10 By the IVT, every $f \in C[0,1]$ with $f(0) < 0 < f(1)$ has a zero. Unfortunately, there is no computable operator for zero-finding working correctly for all $f \in C[0,1]$ with $f(0) < 0 < f(1)$.

However, if we require that f is also increasing, then the operator that returns the unique zero is computable.

Appendix: Full Quote of L. Blum '96

Our perspective is to formulate the laws of computation. Thus, we write not from the point of view of the engineer who looks for a good algorithm which solves the problem at hand, or wishes to design a faster computer. The perspective is more like that of a physicist, trying to understand the laws of scientific computation. [...]

There is a substantial conflict between theoretical computer science and numerical analysis. These two subjects with common goals have grown apart. For example, computer scientists are uneasy with calculus, while numerical analysts thrive on it. On the other hand, numerical analysts see no use for the Turing machine.

The conflict has its roots in another age-old conflict, that between the continuous and the discrete. [...] Algorithms are primarily a means to solve practical problems. There is not even a formal definition of algorithm in the subject [of numerical analysis]. [...] Thus, we view numerical analysis as an eclectic subject with weak foundations; this certainly in no way denies the great achievements through the centuries.