Computer Science for Continuous Data

Survey, Vision, Theory, and Practice of a Computer ~~Algebra~~ *Analysis* System

Franz Brauße[1], Pieter Collins[2], and Martin Ziegler[3](✉)

[1] University of Manchester, Manchester, UK
[2] Maastricht University, Maastricht, The Netherlands
[3] KAIST, Daejeon, Republic of Korea
ziegler@kaist.ac.kr

Abstract. Building on George Boole's work, Logic provides a rigorous foundation for the powerful tools in Computer Science that underlie nowadays ubiquitous processing of discrete data, such as strings or graphs. Concerning continuous data, already Alan Turing had applied "his" machines to formalize and study the processing of real numbers: an aspect of his oeuvre that we transform from theory to practice.

The present essay surveys the state of the art and envisions the future of Computer Science for continuous data: natively, beyond brute-force discretization, based on and guided by and extending classical discrete Computer Science, as bridge between Pure and Applied Mathematics.

1 Introduction and Motivation

Since its early days, Computer Science has enjoyed the support and guidance of Logic, from Theory via Engineering to Practice: recall Alan Turing's 1936 publication preceding nowadays ubiquitous digital computers, or Alonzo Church's Lambda Calculus having led to functional programming languages, or axiomatic structures in Model Theory corresponding to specification of Abstract Data Types, or Hoare Logic for formal program verification—concerning the processing of discrete data, such as graphs or integers or strings.

Continuous data on the other hand commonly arises in Engineering and Science (*natura non facit saltus*) in the form of temperatures and fields; it mathematically includes real numbers, smooth functions, bounded operators, or compact subsets of an abstract metric space. Processing such continuous data has arguably been lacking the foundation and support from Logic in Computer Science that the discrete case is enjoying [11]:

35 years after introduction and hardware standardization of IEEE 754 floating point numbers, mainstream numerics is still governed by this forcible discretization

This work was supported by the National Research Foundation of Korea (grant 2017R1E1A1A03071032) and by the International Research & Development Program of the Korean Ministry of Science and ICT (grant 2016K1A3A7A03950702) and by the European Union's Horizon 2020 MSCA IRSES project #731143.

of continuous data—in spite of violating associative and distributive laws, breaking symmetries, introducing and propagating rounding errors in addition to an involved (and incomplete) axiomatization including NaNs and denormalized numbers.

Deviations between mathematical structures and their hardware counterparts are common also in the discrete realm, such as the wraparound $255 + 1 = 0$ occurring in bytes that led to the "Nuclear Gandhi" programming bug. Therefore nowadays high-level programming languages (like `Java` or `Python`) provide user data types (like `BigInt`) that fully agree with mathematical integers, simulated in software using a variable number of hardware bytes; and advanced discrete data types (such as weighted or labelled graphs) can and do build on that, reliably and efficiently.

The present essay expands on a similar perspective for continuous data types: including real numbers, converging sequences, smooth/integrable functions, bounded operators, compact subsets etc.—exactly, that is, devoid of rounding errors, see Sect. 2. Section 3 discusses imperative programming over such data: with computable semantics including limits, that is, beyond the algebraic realm. Encoding such data over sequences of bits is described in Sect. 4. And Sect. 5 connects discrete complexity theory, with famous classes like $\mathsf{P}/\mathsf{NP}/\#\mathsf{P}/\mathsf{PSPACE}$, to operations on continuous data such as integration. Final steps for putting this theory into practice and its applications are collected in Sect. 6.

2 Computable Continuous Data Types

Data types are at the core of Object-Oriented Programming, see Subsect. 2.1. They constitute Computer Science's counterpart to *structures* in Model Theory. This section explains how, unlike in the discrete case, for continuous data types already their specification often poses a challenge—and may require Kleene Logic (Subsect. 2.2), *enrichment* (Subsect. 2.3) and/or *multivaluedness* (Subsect. 2.4) in order to assert mere computability—before proceeding to complexity questions (Sect. 5). Subsects. 2.5 and 2.6 illustrate these with examples.

2.1 Formal Numerical Software Engineering

Formal software engineering of a data type/object proceeds from (i) problem specification via (ii) algorithm design and (iii) analysis to (iv) proof of optimality and finally (v) implementation and (vi) verification/testing in some high-level object-oriented programming language. Depending on the particular endeavour, some of these stages can of course be kept informal of skipped entirely. Item (iv) here refers to Computational Complexity Theory [49], and implies a (meta) "loop": If the algorithm designed (ii) and analyzed (iii) is not optimal (iv), then start over designing a more efficient one (ii).

Note how (ii) implicitly supposes that the problem specified in (i) actually does admit an algorithmic solution—which in the discrete realm is usually the case. In the real setting, however, any computable function must necessarily be

i) problem specification
ii) algorithm design and
iii) algorithm analysis
iv) proof of optimality — or repeat from (ii)
v) implementation and
vi) verification/testing.

Fig. 1. Six stages of full-fledged Formal Software Engineering. In practice some may be omitted depending on the particular endeavour under consideration.

continuous[1] [63, §2.2+§3.2+§4.3]; hence a naïve problem specification (such as of finding the kernel of a given real matrix) easily results in algorithmic unsolvability. Thus the need arises for another (meta) "loop" in Numerical Software Engineering: if (ii) fails, start over from (i).

2.2 Kleene Logic Data Type, Generalized Sierpiński Topology

The sign function is discontinuous and thus uncomputable. More precisely, a real test like "$x > 0$?" may take more runtime when x is close to zero—and in case $x = 0$ fail to terminate at all:

Fact 1. *Real inequality is "complete" for the Halting Problem $H \subseteq \mathbb{N}$ in the following sense [63, Exercise 4.2.9]:*

a) For every computable real sequence $\bar{x} = (x_j)$, the set $\{j \in \mathbb{N} : x_j > 0\}$ is computably reducible to H.
b) There exists a computable sequence $\bar{x}$ of non-negative real numbers such that said set coincides with H.

Note that fixing a computable real sequence makes the problem independent of encoding issues, as the input consists only of an integer index. The thus non-uniform Item a) follows from the following uniform claim with respect to any of the many equivalent ways of encoding real numbers [63, §4.1]:

c) The partial sign function $\mathrm{sign} : \mathbb{R} \setminus \{0\} \to \{-1, +1\}$ *is computable; same for comparison* $\lessdot : \mathbb{R}^2 \setminus \{(x, x) : x \in \mathbb{N}\} \to \{\mathit{ff}, \mathit{tt}\}$*, where ff and tt denote computational counterparts to Booleans* `TRUE` *and* `FALSE`*.*

A mathematically undefined expression (like 1/0) is sometimes denoted to "have" value $\perp$; comparison "$x > 0$" on the other hand is defined mathematically also in case $x = 0$, but not computationally so. The latter is captured by Kleene Logic $\mathbb{K}$ including, in addition to classical Booleans *tt* and *ff*, as third value *uk* for (mathematically defined but computationally) "unknown". Equip $\mathbb{K}$ with the generalized Sierpiński's topology

$$\big\{\emptyset, \{\mathit{tt}\}, \{\mathit{ff}\}, \{\mathit{ff}, \mathit{tt}\}, \{\mathit{uk}, \mathit{tt}, \mathit{ff}\}\big\}$$

[1] Arguably this also applies to the discrete case, where every function is trivially continuous.

and note that this non-Hausdorff topology fails to separate *uk* from the other elements. Accordingly, a logical expression with mathematical value *uk* fails to evaluate computationally.

$\mathbb{K}$ thus serves as "lazy" data type that can store unevaluated, computationally partial predicates: such as "$x > 0$" for every $x \in \mathbb{R}$, as well as any other promise problem. Recall [19] that a (discrete) promise problem P is a disjoint pair $P^+, P^- \subseteq \mathbb{N}$, such that a query "$m \in P$?" answers *tt* in case $m \in P^+$ and answers *ff* in case $m \in P^-$ and gives no answer *uk* in case $m \notin P^+ \cup P^-$.

2.3 Enrichment/Promises

Promises generalize from decision to function problems, motivated as follows: Topology requires that any non-constant function $f : X \to \mathbb{Z}$ from a connected domain X to the discrete set of integers must be discontinuous. This easily prevents naïve problems from being computable, such as the matrix rank function, or the multiplicities of degenerate eigenvalues. On the other hand providing—in addition to the original continuous data—a suitable integer as input often does render such a problem computable [71]. For example, by Fact 1c), the real sign function is computable on $X_0 := \mathbb{R} \setminus \{0\}$, and on $X_1 := \{0\}$ trivially so. See Subsect. 2.5 below for more examples.

In Constructive Mathematics such an effect is well-known as *enrichment* [36, p. 238/239]; elsewhere also as *advice* [1,8]. It amounts to proceeding from total but discontinuous $f : X \to Y$ to a partial function

$$\tilde{f} :\subseteq X \times \mathbb{Z} \ni (x, k) \mapsto f(x) \in Y \tag{1}$$

for some suitable—and now non-connected—domain $\tilde{X} := \mathrm{dom}(\tilde{f}) \subseteq X \times \mathbb{Z}$ whose projection $X_k := \{x : \exists k : (x, k) \in \tilde{X}\}$ covers X. Put differently, the accompanying argument k entails the *promise* that the "main" input x belongs to the subset $\tilde{X}_k := \{x : (x, k) \in \tilde{X}\}$.

2.4 Multivaluedness/Non-extensionality

Although any computable function must necessarily be continuous, this constraint can be avoided by considering relations, that is, by dropping extensionality. Relations mathematically capture search problems, where a query $x \in X$ has not necessarily one unique answer $y = f(x)$, but a range of possible answers $y \in F(x) \subseteq Y$.

In case the domain X is discrete/countable, namely when arguments $x \in X$ are finitely encoded and read in finite time, then any (deterministic) computation of such a relation F actually computes a *selection*, that is, a function $f \subseteq F$. However in the continuous setting, multivaluedness is well-known unavoidable [42].

Mathematically one may identify the relation F with the single-valued total function $F : X \ni x \mapsto \{y \in Y \mid (x, y) \in F\}$ from X to the powerset 2^Y; but the preferable notation of a *multi*function $f :\subseteq X \rightrightarrows Y$ emphasizes that not every $y \in F(x)$ needs to occur as output. Another important reason to consider multifunctions $f : X \rightrightarrows Y$ distinct from relations $f \subseteq X \times Y$ is related to compactness: Generalizing continuity for single-valued functions, call such a multifunction f *compact-valued* if $f[Z] \subseteq Y$ is compact for every compact $Z \subseteq X$.

A function problem $f : X \to Y$ becomes "easier" when *restricting* arguments to $x \in X'$ for some $X' \subset X$, that is, when proceeding to $f' = f|_{X'}$. A search problem $F : X \rightrightarrows Y$ additionally becomes "easier" when increasing the range of possible answers, that is, when proceeding to some $F' \subseteq X \rightrightarrows Y$ satisfying $F'(x) \supseteq F(x)$ for every $x \in \mathrm{dom}(F')$. Such F' is also called a *restriction* of F. Note that, unlike in the single-valued case, F' need not be a subset of F when considered as graphs.

2.5 Examples

Example 2. The Archimedian Property of real numbers states that, to every $x \in \mathbb{R}$, there exists some $k \in \mathbb{Z}$ with $k \geq x$.

Skolemization yields a function $k : \mathbb{R} \to \mathbb{Z}$ with $\forall x : k(x) \geq x$. The least such function is known as *rounding up* $x \mapsto \lceil x \rceil$ and discontinuous. In fact any such function must be discontinuous and hence uncomputable.

On the other hand the original property formulation suggests formalization as a search (rather than function) problem $\mathrm{Arch} : \mathbb{R} \rightrightarrows \mathbb{Z}$. And indeed this relaxation becomes computable as follows:

Obtain some rational input approximation to the argument $x \in \mathbb{R}$ up to error 2^{-0} and round it up, exploiting that integer fractions can be operated on exactly. Note that a different rational approximation to the same argument x up to error 2^{-0} can yield a different output, i.e., violate extensionality.

Similarly, no integer rounding *function* is computable; whereas the following *multi*function is:

$$\texttt{Round} : \mathbb{R} \ni x \mapsto \{k \in \mathbb{Z} \mid x - 1 < k < x + 1\} \subseteq \mathbb{Z} \tag{2}$$

Example 3. The Fundamental Theorem of Algebra states that, to every monic univariate complex degree-d polynomial $c_0 + c_1 \cdot Z + \cdots + c_{d-1} \cdot Z^{d-1} + Z^d$ decomposes into linear factors $(Z - z_1) \cdots (Z - z_d)$.

This suggests formalization as a mapping

$$F : \mathbb{C}^d \ \ni \ (c_0, \ldots, c_{d-1}) \ \mapsto \ (z_1, \ldots, z_d) \ \in \ \mathbb{C}^d.$$

However note that no order on the roots $z_1, \ldots, z_d$ can be imposed mathematically; hence F should naturally be considered as multifunction.

Moreover it turns out that, similarly to Example 2, multivalued F is computable while no single-valued selection of F is [59].

Example 4. Consider the problem of computing a basis of the kernel of a real matrix A given by its entries. Note that this problem is already multivalued problem to begin with, since such a basis is usually far from unique.

Gaussian elimination involves pivot search and thus tests for real inequality—which are uncomputable: recall Fact 1. Indeed already the (unique) cardinality of a (non-unique) basis depends discontinuously on the matrix entries.

However enriching input A with said cardinality $= \operatorname{rank}(A) \in \mathbb{N}$ does render such a basis computable [71].

Example 5. Consider the problem of computing the spectral decomposition, formalized as computing an (!) eigenvector basis, to a given symmetric real matrix $A \in \mathbb{R}^{d\times d}$.

According to Example 3, a tuple of eigenvalues, repeated according to their multiplicities, can be computed via the characteristic polynomial. However the integer-valued multiplicities themselves are discontinuous and uncomputable in their dependence on the matrix entries. Moreover computing an eigenvector basis, although non-unique/multivalued, is impossible; whereas enriching input A with the number $k \in \{1,\ldots,d\}$ of distinct eigenvalues renders the problem computable [71, Theorem 11]. Moreover such d-fold advice turns out as optimal [70, Theorem 46].

We remark that computing eigen*spaces* is possible, when equipping the latter with the right topology.

Example 6. Alternative to the partial sign function from Fact 1, the following total but multivalued so-called *soft* test [67, §6] is computable as well [7, p. 491]:

$$\text{“}x <_n 0'' \quad = \quad \begin{array}{ll} tt & \text{in case } x < -2^{-n}, \\ f\!f & \text{in case } x > 2^{-n}, \text{ and} \\ \text{either } tt \text{ or } f\!f & \text{in case } -2^{-n} \le x \le 2^{-n}. \end{array} \tag{3}$$

Example 7. Fix promise problems $P_0, P_1, \ldots, P_{d-1} \subseteq \mathbb{N}$ with the aforementioned computational semantics “$(m \in P_j) \in \mathbb{K}$”. Consider the partial multi-valued mapping $\mathsf{choose} : \mathbb{K}^d \rightrightarrows \{0,1,\ldots,d-1\}$ assigning to $\big((m \in P_0),\ldots,(m \in P_{d-1})\big)$ to some j such that $(m \in P_j) = tt$. This is computable!

2.6 More Continuous Data Types

Real numbers are arguably the most basic continuous structure in Calculus; vector, sequence, and function spaces for instance build on top of them. Similarly, having turned real numbers into a computable data type (“level 0”) enables now turning more advanced spaces from Mathematics into computable ones—using the aforementioned techniques to deal with discontinuities: enrichment (Subsect. 2.3) and multivaluedness (Subsect. 2.4). Specifically polynomials and matrix operations have been discussed in Subsect. 2.5 above. Note that each such object can be described with finitely many real numbers: “level 1”.

2) Sequence spaces ℓ^p are the next level, each element consisting of countably infinitely many real numbers. A plethora of investigations [9,34,43,45] provide guidance on suitable enrichment (such as integer bounds on the norm) to turn them into computable data types.

3) Power series can be identified with their germs/coefficient sequences in appropriately enriched sequence spaces [28, §3.1]; and analytic functions are local power series—of which finitely many suffice to “cover” any fixed compact subset of their domain [28, §3.2].
4) The hyperspace of non-empty compact subsets of Euclidean space is computably closed under union and under image of continuous functions [63, §6.2], but not under (even promised non-empty) intersection [63, Exercise 5.1.15]. The hyperspace of *convex* compact subsets does satisfy this, and additional, computational closure properties [39].
5) Space of probability measures [24,44,57] and subspace of Haar measures on compact groups [51].
6) Spaces of continuous [63, §6.1], of smooth [28], and of integrable functions [29, 60]; equipped with operations like (anti or weak) derivative, or trace.
7) Differential geometry, that is, the hyperspace of closed (smooth) manifolds equipped with (smooth) tensor fields on them.

3 New Numerical Programming

Since the early days of automated digital processing in assembly code, programming has made tremendous progress. Nowadays high-level languages provide both convenience/intuition and soundness/reliability—regarding discrete data, such as integers or strings.

Numerical programming differs from this classical realm in that the underlying data type intrinsically incurs errors, namely from rounding. Tracing and bounding the propagation of such deviations is up to the user programmer, and the involved IEEE 754 standard makes reliable coding inconvenient. Practitioners therefore often imagine operating on real (instead of floating point) numbers. This implicit approach thus trades convenience for reliability.

Algebraic numbers can be processed exactly, and the formal verification of algebraic programs [48] may build on Tarski’s decidability of the First-Order Theory of this algebraically closed field (although the latter excludes the exponential as well as many other important analytic functions in Science and Engineering and Calculus [6]).

Computable Analysis [63] on the other hand does provide a realistic characterization of un/computable real functions beyond the algebraic realm. It is however based on the (type-2) Turing machine model: theoretically important but practically inconvenient for programming, not to mention formal verification.

Subsection 3.1 recalls an equivalent but convenient imperative model of computation called ERC that comes as close to, and thus provides a sound formalization of, common implicit conceptions underlying numerical programming. Previous and future ways for implementing it on actual digital computers are discussed in Subsect. 3.2. ERC modifies the semantics of real comparison, and Subsection 3.3 illustrates its use with some basic example algorithms. Subsection 3.4 provides a road map of continuous data types to next implement in ERC.

3.1 Analytic Programming

The preprint [10] formalizes a (proof-of-concept) imperative programming language called `ERC` supporting a data type `REAL` that agrees with the mathematical structure $\mathbb{R}$, exactly. Its semantics is carefully designed to capture common conceptions (sometimes implicitly) underlying numerical coding, while achieving "Turing-completeness" over the reals: Any function *realizable* in ERC is computable in the sense of Computable Analysis—and vice versa. ERC thus combines the structural benefits of Computable Analysis (such as closure under composition and including transcendental functions) with the intuitive convenience and practical pervasion of object-oriented imperative programming to replace the hassles of Turing machines.

Paradigm 8. *A mathematical partial function $f :\subseteq \mathbb{R} \to \mathbb{R}$ is* realized *in* ERC *as a (multi-)function of type $\mathbb{Z} \times \mathbb{R} \rightrightarrows \mathbb{R}$: It receives its real argument x exactly, as well as a separate integer*[2] *parameter $p \to -\infty$, and must eventually return some approximation to $y = f(x)$ up to absolute error $2^p \to 0$. To this end during intermediate calculations, it may use arithmetic operations free of rounding errors. The "result" of a possibly partial comparison "$x > y$" according to Fact 1(c) can be stored in a logic variable of type* `KLEENEAN` *(Subsect. 2.2), and can be evaluated safely using the multivalued operation from Example 7 to yield a total program.*

The discrepancy between exact argument and approximate return value might suspect to void closure under composition [68, p. 325]; however in combination with the modified partial semantics of comparison (Fact 1c), Computable Analysis does assert closure under composition [64]. In particular an ERC program expressing real function f as above may, in addition to using arithmetic operations, call another ERC representing some other real function g with exact argument z to receive and continue processing real return value $w = g(z)$ exactly, i.e., without having to worry about error propagation.

Remark 9. The error bound 2^p ($\mathbb{Z} \ni p \to -\infty$) is preferable over, say, $1/|p|$:

1. It reflects that π has been approximated up to $2^{-\text{billions}}$ [26].
2. It renders the underlying logic decidable; see Subsect. 3.5.
3. It yields numerical characterizations of popular discrete complexity classes; see Subsect. 5.1.

A similar but more practically fleshed-out programming language for continuous data has been devised in [3].

3.2 Implementations

Superficially, operating on real (including transcendental) numbers exactly as postulated by ERC might seem technically infeasible: within finite time, only

[2] One could replace it with some real error bound $\varepsilon > 0$.

finite information can be processed. This reproach is valid in operational semantics, where (the order of) user commands correspond to finite blocks of machine instructions executed in the same order. However functional programming regularly removes this implicit condition, and adapting that relaxation to the imperative setting does enable the above semantics in user space [38,46,69].

Technically speaking, operations can be realized "exactly" by actually processing approximations of variable but finite precision; precision chosen automatically, for any given n, such that (a) the output accuracy attained after initial and propagated errors is $\leq 1/2^n$, and (b) the program flow remains indistinguishable from hypothetical exact calculations.

Both conditions can be implemented by object-oriented overloading the operations involving continuous data. One approach executes the user program symbolically, recording expressions of all variables' contents in dependence on the initial arguments—expressions which thus can be evaluated in any finite initial precision and with error propagation whenever needed for (a) output or (b) to decide/branch on a comparison between two distinct reals. Another approach trades runtime for memory namely, instead of recording and silently reevaluating symbolic expressions, silently re-executes the user program repeatedly in increasing but finite initial precision until sufficient to (b) decide all comparisons/branches and (a) attain the desired output accuracy. Note that either approach requires the user program to be devoid of side effects, as common in functional programming and "desirable" (but now mandatory) in the imperative setting.

3.3 Example ERC Programs

Paradigm 8 was designed to yield a rigorous and real Turing-complete semantics of analytic programming closest to numerical intuition. Thus it requires only little adaptation to replace classical but uncomputable tests with their replacements, as exemplified in this subsection:

Example 10. The Soft Test $x <_n 0$ from Example 6 can be expressed in ERC:

$$\mathsf{choose}\left(x > 2^{-n+1} \;,\; x < 2^{-n} \right) \mathit{ieq}\; 1$$

Example 11. The multivalued rounding "function" from Eq. 2 after Example 2 can be represented in ERC as follows:

```
INTEGER Round(x:REAL);
LET k:INTEGER=0;
WHILE choose( x<1 , x>1/2 ) = 1
  DO  k:=k+1;  x:=x-1;  ENDWHILE;
WHILE choose( x>-1 , x<-1/2 ) = 1
  DO  k:=k-1;  x:=x+1;  ENDWHILE;
RETURN k;
```

Note that the loop bodies are executed roughly $|x|$ times, that is, exponential in the binary length of the output. See [10, §2.5] for a more efficient version sufficing with a linear number of loop iterations.

Classically, bisection is employed for finding the root of a function $f : [a; b] \to \mathbb{R}$. Rigorously speaking however, this may fail in case the chosen mid-point $c = (a + b)/2$ already happens to coincide with the root of the function: In this case $f(c) = 0$, the test of sign $f(c)$ to decide which of the two sub-intervals to proceed with, fails to terminate; recall Fact 1. Instead, *tri*section with overlapping sign conditions has been suggested [20, p. 336] and can be represented in ERC as follows:

Example 12. `Program` Trisection(p : `INTEGER`, f : `REAL` $\to$ `REAL`)

```
let $a$ : REAL $= 0$;  let $b$ : REAL $= 1$;
while choose$\big(\imath(p) \gtrdot b - a,\ b - a \gtrdot \imath(p-1)\big) = 1$
    let $a'$ : REAL $= b/3 + 2 \times a/3$;   let $b'$ : REAL $= 2 \times b/3 + a/3$;
    if choose$\big(0 \gtrdot f(a') \times f(b)\,,\ 0 \gtrdot f(a) \times f(b')\big) = 1$
        then $b := b'$   else$a := a'$ end if;
end while;  return a
```

Gaussian Elimination, as reported in Example 4, requires and suffices with receiving the rank of the given matrix as additional input in order to become computable. A rigorous implementation using full pivoting can be found at http://github.com/realcomputation/iRRAMx.

Example 13. Common implementations drawing the Mandelbrot Set lack reliability for two reasons: First the defining iteration is conducted in floating-point arithmetic with rounding and truncation errors that propagate and make the computed sequence differ from the mathematical one. Secondly, said sequence is calculated for a fixed number N of iterations after which it is considered to not diverge—without actual mathematical justification.

The first rigorous algorithm for computing the Mandelbrot Set (subject to the Hyperbolicity Conjecture) is due to Peter Hertling [21] with first implementation due to Jihoon Hyun:
http://github.com/realcomputation/MANDELBROT.

3.4 Advanced and Upcoming ERC Programs

Some of the computable continuous data types from Subsects. 2.5 and 2.6 already have been implemented in ERC: such as polynomial root-finding (in the sense of Example 3) with QR-algorithm and Wilkinson Shift, or matrix Gaussian Elimination (in the sense of Example 4) with full pivoting [50], or analytic functions [58] or compact metric groups [51] or solution operators to selected PDEs [35].

Future efforts are directed towards similarly implementing:

- the hyperspace of compact subsets of Euclidean space [25];
- spaces of integrable functions defined on a compact Euclidean domain;
- closed manifolds from differential geometry;
- Random sampling of continuous objects (Subsect. 5.2).

See also Subsect. 6.1 below.

3.5 Verification/Testing

According to Murphy's Law of Computing, "Every non-trivial program has at least one bug". Verification and Testing are two major approaches to prevent or at least reduce the number of errors. Beyond heuristical success stories [55], both methods arguably lack logical justification: A program may well work correctly throughout years and on billions of practical test instances yet still contain fundamental flaws, such as *zero-day exploits*. And Gödel Incompleteness/Hilbert's Tenth Problem translates to arithmetic programs which are correct, but whose correctness provably cannot be proven [13].

Relative to such constraints on integer processing, formally verifying the correctness of floating point calculations is possible [4]—but involved and messy: Designed by the Institute of Electrical and Electronics Engineers, the IEEE 754 standard revolves around 1980ies hardware capabilities to support for example NaNs and de/normalized numbers while violating mathematical associate and distributive laws as well as logical completeness.

Real closed fields of characteristic zero on the other hand do admit an intuitive and elegant axiomatization[3], which is furthermore logically complete according to Tarski. More generally, consider the many-sorted structure involving real numbers and Presburger integers connected via the "precision" embedding $\imath : \mathbb{Z} \ni z \mapsto 2^z \in \mathbb{R}$: It can express many (local) properties of ERC programs and, building on work by van den Dries [2,14], is decidable: justifying automatic formal verification. (Whereas other embeddings, such as $\tilde{\imath} : \mathbb{N} \setminus \{0\} \ni n \mapsto 1/n \in \mathbb{R}$, lead to Gödel undecidability.) Recall Remark 9 and see §4.1 in arXiv:1608.05787 for details.

4 Coding Theory

Digital computers and Turing machines naturally operate on sequences of bits; processing any other data, such as integers or graphs, needs first fixing encodings for input and output. In the discrete case, this is usually straightforward and/or complexity-theoretically inessential (up to polynomial time, say). However concerning continuous data, already real numbers suggest various encodings with surprisingly different algorithmic properties: ranging from the computably "unreasonable" binary expansion via qualitatively to polynomially and even linearly complexity-theoretically "reasonable" signed-digit expansion. But how to distinguish between un/suitable encodings of other spaces common in Calculus and Numerics, such as Sobolev?

This (meta) question has long been answered regarding qualitative computability: *admissibility* [37,56] is a crucial condition for an encoding of a space X to be "reasonable". Following this conception, encodings are partial surjective mappings (historically called *representations*) from Cantor space onto X; and said mapping is required to be (a) sequentially continuous and (b) maximal with respect to sequentially continuous reduction [63, §3.2]. Admissible encodings are

[3] Without the second-order property of being topologically complete.

guaranteed to exist for a large class of topological spaces, and to be Cartesian closed. And for (precisely) these does the sometimes so-called *Main Theorem* hold: which characterizes continuity of functions by the continuity of mappings translating codes, so-called *realizers*.

Subsection 4.1 summarizes the preprint [41] on quantitatively/complexity-theoretically refining said qualitative/computable admissibility from topological to metric spaces. Further tailoring such efficient encoding to spaces with additional structure is discussed in Subsect. 4.2.

4.1 Quantitative Coding Theory of Compact Metric Spaces

[41] develops a generic approach to refine qualitative computability over topological spaces to quantitative complexity over metric spaces. It strengthens the notion of unqualified admissibility to *polynomial* and to *linear admissibility*. Informally speaking, the latter two require a representation to be (a) almost "optimally" continuous (namely linearly/polynomially *relative* to the space's entropy) and (b) maximal with respect to *relatively* linearly/polynomially continuous reductions.

A large class of spaces is shown to admit a quantitatively admissible representation, including a generalization of the signed-digit encoding; and quantitatively admissible representations exhibit a quantitative strengthening of the qualitative *Main Theorem*, namely now characterizing quantitative continuity of functions by quantitative continuity of realizers. Quantitative admissibility thus provides the desired criterion for complexity-theoretically "reasonable" encodings.

The contribution then rephrases quantitative admissibility as quantitative continuity of both the representation and of its set-valued inverse. For the latter purpose, it adapts from [52] a new notion of sequential continuity for multifunctions. By establishing a quantitative continuous selection theorem for multifunctions between compact ultrametric spaces, it extends the above quantitative *Main Theorem* from functions to multifunctions aka search problems. Higher-type complexity is captured by generalizing Cantor's (and Baire's) ground space for encodings to other (compact) *ultra*metric spaces.

4.2 Encoding Advanced Spaces in Analysis

Structures expanding on compact metric spaces support operations beyond the metric. Making also these computable (subject to admissible encodings) is discussed in Subsect. 2.6 above. Minimizing their computational cost in turn relies on further refining the encodings from Subsect. 4.1: Work in progress develops and compares tailored representations for spaces of (say, square) integrable functions, and for Sobolev spaces of weakly differentiable functions. Such spaces underlie the mathematical theory of partial differential equations [62], and are thus required for the following complexity considerations:

5 Complexity Theory of Continuous Data

Over the past decades, Numerics has devised a myriad of methods: for efficiently computing algebraic and transcendental constants and functions, for solving ordinary and differential and partial differential equations, for optimization under constraints etc.

The efficiency of such a method can often be shown optimal by comparison to the quantitative stability of the problem it solves. When small perturbations of the input lead to large changes in output, algorithms must necessarily process and operate on high-precision data, incurring a large number of bit manipulations.

Function maximization and Riemann integration are stable; yet information-theoretically their approximation up to guaranteed absolute error $1/2^n$ depends on exponentially many sample points already in the smooth case; recall Remark 9. This demonstrates information theory as another method for rigorous lower complexity bounds in Numerics.

But what if the function is fixed, so that only the precision parameter $n \in \mathbb{N}$ remains as input? Subsect. 5.1 reports on surprising connections of numerical problems in this setting to *unary* classical (i.e., discrete) complexity classes. Subsection 5.2 addresses the question of adapting randomization from the discrete to the continuous setting.

5.1 Computational Complexity of Continuous Data

For any fixed polynomial-time computable real function $f : [0; 1] \to \mathbb{R}$, Harvey Friedman and Ker-I Ko had observed that its maximum $\max(f)$ can be computed relative to an NP_1 oracle and its definite integral $\int f$ relative to a $\#\mathsf{P}_1$ oracle [17,31,33]. Recall that the latter denote restrictions of the famous complexity classes NP and $\#\mathsf{P}$ to inputs $n \in \mathbb{N}$ encoded in unary; and maximizing and integrating/counting a *given, discrete* (e.g., Boolean) function are well-known complete for these respective classes.

Parametric maximization is the problem of maximizing $f : [0; 1] \to \mathbb{R}$ not on the entire interval, but on the subinterval $[0; x]$ for a given real number $x \leq 1$. Similarly, computing the indefinite integral $\int_0^x f(y)\, dy$ involves two arguments: real x and integer precision parameter n. These turn out to be computable in polynomial time relative to NP oracles and $\#\mathsf{P}$ oracles, respectively. And, perhaps surprisingly, this is optimal: there exist (even smooth) polynomial-time computable real functions such that polynomial-time algorithms for parametric maximization and indefinite integration yields polynomial-time solutions to NP and $\#\mathsf{P}$, respectively [32].

Thus, perhaps contrary to intuition, proceeding from discrete to smooth instances does not help (enough) in the rigorous sense of computational complexity theory to proceed from $\mathsf{NP}/\#\mathsf{P}$ to polynomial time.

$\#\mathsf{P}_1$-"completeness" of definite integration generalizes from the real unit interval with respect to the Lebesgue measure to a large class of compact metric groups with respect the Haar measure [51]. Akitoshi Kawamura's 2010 breakthrough result [27] similarly characterizes PSPACE via solving 1D smooth ordinary differential equations. And recent contributions relate #P to solving two linear prototype PDEs, namely (elliptic) Poisson [30] and (parabolic) Heat Equation [35].

Next up on the to-do list is a complexity-theoretic classification of the (hyperbolic) linear Wave Equation, and of the non-linear Navier-Stokes Equation. Subject the Millennium Prize Problem, Navier-Stokes maintains regularity and its solutions remain in classical spaces of continuously differentiable functions with their established coding and computability and complexity theory [61]. But regarding the Wave Equation, its regularity theory is well-established to require Sobolev spaces for computability investigations [53,66]; and Subsect. 4.2 develops the quantitative coding theory necessary for complexity considerations.

5.2 Algorithmic Random Sampling of Continuous Data

Monte Carlo algorithms date back to the Manhattan Project, and randomization has since evolved into an important technique in Computer Science: building up from random bits to random integers, random real numbers etc. Like every subset of natural numbers giving rise to a decision problem, every probability measure gives rise to three conceptually distinct computational problems: (a) evaluating, (b) integrating, and (c) random sampling.

Under mild assumptions, evaluation (a) and integration (b) are known computably equivalent [65]; see also the many works of Hoyrup. The general relation of (a) and (b) to (c) random sampling however seems open so far regarding computability. This includes generalizing the real case with Lebesgue measure to other Haar measures on compact groups, cmp. Subsect. 2.6. More generally consider the problem of computably sampling elements from a separable but not necessarily (sigma-)compact space, such as the Wiener space [40].

Following computability, the natural next question is concerned with computational complexity. Recall Subsect. 5.1 that, over the reals, (iii) sampling takes polynomial time while (ii) integrating characterizes #P. Both generalize from the real unit interval to convex bodies in Euclidean space [12,15]. Beyond the continuous Lebesgue measure, integration remains #P-hard for singular measures [16] but becomes algorithmically easy for discrete (e.g., Dirac) measures.

Question 14. Is there a probability measure space where sampling is significantly harder than integration?

6 From Theory to Applications via Practice

Sections 2 to 5 have expanded on four central concepts from classical computer science and how to extend them to the continuous setting. Key examples illustrate how this has been achieved or is currently in progress or what to approach next: to provide proofs-of-concept and opportunities to gather experience and guidance, to pave the path. The present section explores and details ways to finally flesh out between and beyond said case studies, to turn the theory into practice, and to pursue applications.

The present section explores and details ways for finally fleshing out between and beyond said case studies, to turn the theory into practice, and to pursue applications: Subsect. 6.1 envisions thus growing a rich software library of continuous data types. The question of inputting and outputting smooth vector fields is addressed in Subsect. 6.3. Combining such human-computer interface with the software library for *analytical* computing complements common Computer *Algebra* Systems, see Subsect. 6.4. Subsection 6.5 promotes its benefits to Experimental Mathematics.

6.1 Software Library

The rigorous paradigm of Analytic Programming (Sect. 3) finally allows to extend the six stages (i)–(vi) of Formal Software Engineering (Subsect. 2.1) from discrete to continuous problems. It thus enables creating a collection of abstract data types that build up from basic real numbers to the structures of Advanced Calculus, reliably. Above we have illustrated selected stages of this process (Fig. 1) with independent examples, such as: specification (Subsect. 2.6), efficient coding (Sect. 4), complexity (Subsect. 5.1), imperative implementation (Subsect. 3.4), and verification/testing (Subsecti. 3.5).

After completing these proofs-of-concepts comes extending, for each of the above case-study data type levels (0) and (1) from Subsect. 2.6, the example stage from the demonstration to range full-stack from (i) to (vi); and then similarly applying formal Numerical Software Engineering from specification (i) to verification/testing (vi) for the advanced structures in Calculus on levels (2) to (7). This yields a gradually growing collection of reliable data types with algorithmically optimal methods in agreement with constructive proofs. Specifically the following four examples, formulated abstractly and generically in Mathematics, translate equally universally to algorithms using overloading:

Example 15. Equations are often solved by means of iterations:

a) The multiplicative inverse $y = x^{-1}$ can be computed as solution to $1/y - x = 0$ by Newton's method $y_{n+1} = y_n \cdot (2 - x \cdot y_n)$: generically in many rings, such as for example of matrices or operators.
b) Similarly, the square root $y = \sqrt{x}$ as solution of the equation $x^2 = y$ is also often computed by means of Newton iterations aka Babylonian method $y_{n+1} = \big(y_n + x \cdot y_n^{-1}\big)/2$: again generically in many rings.
c) Picard's method for solving ODEs amounts to iterations according to Banach's Fixedpoint Theorem in a suitable space of smooth functions.
d) Solutions to Navier-Stokes' nonlinear PDE are also mathematically shown to exist [18, §2] and being computable [61] by means of iterations in some space of integrable functions.

6.2 Hardware Acceleration

Being Turing-complete over the reals, Analytic Programming (Sect. 3) hides but must and can build on processing ordinary variable precision approximations. IEEE 754 floating point numbers have fixed precision, but enjoy a constant-factor acceleration from hardware support—compared to software solutions. Similarly accelerating ERC will thus combine the best of both worlds: reliability and efficiency.

Previous work has already managed to beat the highly optimized software library `MPFR`in quadruple precision by instead operating on pairs of hardware `doubles` [23,25]. Alternative approaches may explore SIMD parallel processing of many single precision floating point numbers on a GPU; or may develop dedicated FPGAs/ASICs for multiprecision processing. In both cases, the communication bottleneck to the main CPU/memory requires outsourcing complex operations and sequences on once transferred data.

This endeavour naturally proceeds with the support of and collaboration with Electrical Engineering.

6.3 User Interface

Processing is the middle part of the IPO model, whose extension from discrete to continuous data have been discussed above. The first and last part of IPO refer to input and output. Historical human-computer-interfaces like keyboard/printer can input/output symbolic data, and are thus suitable for Computer Algebra Systems manipulating expressions: one way of representing functions, but lacking intuition. Intuitively and interactively "grabbing" and "pulling" is supported by common graphical user interfaces, based on mouse devices for input and monitors for output—but these are limited to 2D.

VR glasses can visualize 3D, but doing so for opaque non-scalar fields is challenging to put it mildly; and motion sensing game controllers (like Kinect or Nintendo Switch Pro) allow for "grabbing" and "pulling" in 3D, but they do not support "twisting", i.e., they cannot edit vortices.

Thus arises the need to develop a user interface for input and output of real functions "living" in higher dimensions, such as scalar (e.g., temperature) fields in 2D and 3D, or vector (e.g., force) fields. Its core challenge is for a haptic data glove that, conversely to detecting user motions in space (as mentioned already supported by existing models), can also *exercise* free forces, i.e., to pull/drag the user's hand in any direction and magnitude: allowing to "feel" (as opposed/complement to "view" in VR) vector fields. Moreover, in order to both feel and modify vortices of vector fields, the glove will be able to both sense and exercise twisting motions according to any rotation vector.

This endeavour naturally proceeds with the support of and collaboration with Mechanical Engineering.

6.4 Computer *Analysis* System

Combining the software library from Subsect. 6.1 with an interactive user interface (cmp. Subsect. 6.3) yields a Computer *Analysis* System: complementing contemporary Computer *Algebra* Systems, either standalone or—preferably—as seamless extension to a suitable open system like `OSCAR`. Here each abstract data type naturally turns into a *package* (interface). The plan is for further integration with some theorem proof assistant, such as Coq/HOL.

6.5 Experimental Transcendental Mathematics

The rise of Computer *Algebra* Systems has truly boosted experimental approaches to discrete branches of Mathematics; see for instance the works of Shalosh B. Ekhad. Computer-assisted proofs of statements in continuous Mathematics are a rising field, with breakthroughs concerning for example the Kepler Conjecture or Smale's 14th Problem. But these contributions remain isolated, with each approach computationally tailored (e.g., whether using hardware floats, or `MPFR`, and at which precision) to the particular problem: challenging for good reasons [5,47], and far from the convenience and turnkey approaches available in the discrete realm.

The software library from Subsect. 6.1 will remedy this deficiency, supporting reliable off-the-shelf computations for example in transcendental number theory: by putting a variety of theoretical algorithms into practice [22,54] and by spurring the development of new ones.

References

1. Ambos-Spies, K., Brandt, U., Ziegler, M.: Real benefit of promises and advice. In: Bonizzoni, P., Brattka, V., Löwe, B. (eds.) CiE 2013. LNCS, vol. 7921, pp. 1–11. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39053-1_1
2. Avigad, J., Yin, Y.: Quantifier elimination for the reals with a predicate for the powers of two. Theor. Comput. Sci. **370**(1–3), 48–59 (2007). https://doi.org/10.1016/j.tcs.2006.10.005
3. Bauer, A.: Clerical. https://github.com/andrejbauer/clerical (2017)
4. Boldo, S., Jourdan, J.H., Leroy, X., Melquiond, G.: Verified compilation of floating-point computations. J. Autom. Reason. **54**(2), 135–163 (2015)
5. Bornemann, F., Laurie, D., Wagon, S., Waldvogel, J.: The SIAM 100-Digit Challenge. SIAM (2004). http://www.siam.org/books/100digitchallenge/
6. Brattka, V.: The emperor's new recursiveness: the epigraph of the exponential function in two models of computability. In: Ito, M., Imaoka, T. (eds.) Words, Languages & Combinatorics III, pp. 63–72. World Scientific Publishing, Singapore (2003), iCWLC 2000, Kyoto, Japan, March 14–18 (2000)
7. Brattka, V., Hertling, P.: Feasible real random access machines. J. Complex. **14**(4), 490–526 (1998)
8. Brattka, V., Pauly, A.: Computation with advice. In: Zheng, X., Zhong, N. (eds.) CCA 2010, Proceedings of the Seventh International Conference on Computability and Complexity in Analysis. Electronic Proceedings in Theoretical Computer Science, vol. 24, pp. 41–55 (2010)

9. Brattka, V., Schröder, M.: Computing with sequences, weak topologies and the axiom of choice. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 462–476. Springer, Heidelberg (2005). https://doi.org/10.1007/11538363_32
10. Brauße, F., et al.: Semantics, logic, and verification of “exact real computation”. Tech. rep., arXiv (2021)
11. Braverman, M., Cook, S.A.: Computing over the reals: foundations for scientific computing. Notice AMS **53**(3), 318–329 (2006)
12. Cho, J., Park, S., Ziegler, M.: Computing periods . . .. In: Proceedings of the WALCOM: Algorithms and Computation - 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, 3–5 March 2018, pp. 132–143 (2018). https://doi.org/10.1007/978-3-319-75172-6_12
13. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM J. Comput. **7**(1), 70–90 (1978). https://doi.org/10.1137/0207005
14. Dries, L.v.d.: The field of reals with a predicate for the powers of two. Manus. Math.**54**, 187–196 (1986), http://eudml.org/doc/155108
15. Dyer, M.E., Frieze, A.M., Kannan, R.: A random polynomial time algorithm for approximating the volume of convex bodies. J. ACM **38**(1), 1–17 (1991). https://doi.org/10.1145/102782.102783
16. Férée, H., Ziegler, M.: On the computational complexity of positive linear functionals on $\mathcal{C}[0;1]$. In: Kotsireas, I.S., Rump, S.M., Yap, C.K. (eds.) MACIS 2015. LNCS, vol. 9582, pp. 489–504. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32859-1_42
17. Friedman, H.: The computational complexity of maximization and integration. Adv. Math. **53**, 80–98 (1984). https://doi.org/10.1016/0001-8708(84)90019-7
18. Giga, Y., Miyakawa, T.: Solutions in l^r of the Navier-stokes initial value problem. Arch. Ration. Mech. Anal. **89**(3), 267–281 (1985)
19. Goldreich, O.: On promise problems: a survey. In: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.) Theoretical Computer Science. LNCS, vol. 3895, pp. 254–290. Springer, Heidelberg (2006). https://doi.org/10.1007/11685654_12
20. Hertling, P.: Topological complexity with continuous operations. J. Complex. **12**, 315–338 (1996). https://doi.org/10.1006/jcom.1996.0021
21. Hertling, P.: Is the Mandelbrot set computable? Math. Log. Q. **51**(1), 5–18 (2005)
22. Hertling, P., Spandl, C.: Computing a solution of Feigenbaum’s functional equation in polynomial time. Log. Methods Comput. Sci. **10**(4), 4:7, 9 (2014). https://doi.org/10.2168/LMCS-10(4:7)2014
23. Hida, Y., Li, X.S., Bailey, D.H.: Library for double-double and quad-double arithmetic. Tech. rep, Lawrence Berkeley National Laboratory (2007)
24. Hoyrup, M., Rute, J.: Computable measure theory and algorithmic randomness. In: Handbook of Computability and Complexity in Analysis. TAC, pp. 227–270. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-59234-9_7
25. Jiman, H.: Real computation: from computability via efficiency to practice. M.sc. thesis, School of Computing (2021)
26. Kanada, Y.J.: 計算機による円周率計算 (特集 円周率 π). Math. Cult. **1**(1), 72–83 (2003). Calculation of circumferential ratio by computer
27. Kawamura, A.: Lipschitz continuous ordinary differential equations are polynomial-space complete. Comput. Complex. **19**(2), 305–332 (2010). https://doi.org/10.1007/s00037-010-0286-0
28. Kawamura, A., Müller, N., Rösnick, C., Ziegler, M.: Computational benefit of smoothness: parameterized bit-complexity of numerical operators on analytic functions and Gevrey’s hierarchy. J. Complex. **31**(5), 689–714 (2015). https://doi.org/10.1016/j.jco.2015.05.001

29. Kawamura, A., Steinberg, F., Ziegler, M.: Towards computational complexity theory on advanced function spaces in analysis. In: Beckmann, A., Bienvenu, L., Jonoska, N. (eds.) CiE 2016. LNCS, vol. 9709, pp. 142–152. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40189-8_15
30. Kawamura, A., Steinberg, F., Ziegler, M.: On the computational complexity of the Dirichlet problem for Poisson's equation. Math. Struct. Comput. Sci. **27**(8), 1437–1465 (2017). https://doi.org/10.1017/S096012951600013X
31. Ko, K.I.: The maximum value problem and NP real numbers. J. Comput. Syst. Sci. **24**, 15–35 (1982)
32. Ko, K.I.: Complex. Theory Real Funct. Progress in Theoretical Computer Science, Birkhäuser, Boston (1991)
33. Ko, K.I., Friedman, H.: Computational complexity of real functions. Theoret. Comput. Sci. **20**, 323–352 (1982)
34. Køber, P.K.: Uniform domain representations of ℓ_p-spaces. Math. Log. Q. **180**(2), 180–205 (2007)
35. Koswara, I., Pogudin, G., Selivanova, S., Ziegler, M.: Bit-complexity of solving systems of linear evolutionary partial differential equations. In: Santhanam, R., Musatov, D. (eds.) CSR 2021. LNCS, vol. 12730, pp. 223–241. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79416-3_13
36. Kreisel, G., Macintyre, A.: Constructive logic versus algebraization, I. In: Troelstra, A., van Dalen, D. (eds.) The L. E. J. Brouwer Centenary Sympos. Studies in Logic and the Foundations of Mathematics, vol. 110, pp. 217–260. North-Holland, Amsterdam (1982), (Noordwijkerhout, June 8–13 1981)
37. Kreitz, C., Weihrauch, K.: Theory of representations. Theoret. Comput. Sci. **38**, 35–53 (1985)
38. Lambov, B.: RealLib: an efficient implementation of exact real arithmetic. Math. Struct. Comput. Sci. **17**, 81–98 (2007)
39. Le Roux, S., Ziegler, M.: Singular coverings and non-uniform notions of closed set computability. In: Dillhage, R., Grubba, T., Sorbi, A., Weihrauch, K., Zhong, N. (eds.) Proceedings of the Fourth International Conference on Computability and Complexity in Analysis (CCA 2007). Electronic Notes in Theoretical Computer Science, vol. 202, pp. 73–88. Elsevier (2008), CCA 2007, Siena, Italy, 6–18 June 2007
40. Lee, H.: Random sampling of continuous objects. Ph.D. thesis, School of Computing (2020)
41. Lim, D., Ziegler, M.: Quantitative coding and complexity theory of continuous data. Tech. rep., arXiv (2021)
42. Luckhardt, H.: A fundamental effect in computations on real numbers. Theoret. Comput. Sci. **5**(3), 321–324 (1977)
43. McNicholl, T.H.: A note on the computable categoricity of ℓ^p spaces. In: Beckmann, A., Mitrana, V., Soskova, M. (eds.) CiE 2015. LNCS, vol. 9136, pp. 268–275. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20028-6_27
44. Mori, T., Tsujii, Y., Yasugi, M.: Computability of probability distributions and characteristic functions. Log. Methods Comput. Sci. **9**, 3:9, 11 (2013). https://doi.org/10.2168/LMCS-9(3:9)2013
45. Mostowski, A.: On computable sequences. Fundam. Math. **44**, 37–51 (1957)
46. Müller, N.T.: The iRRAM: exact arithmetic in C++. In: Blanck, J., Brattka, V., Hertling, P. (eds.) CCA 2000. LNCS, vol. 2064, pp. 222–252. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45335-0_14

47. Nakao, M.T., Plum, M., Watanabe, Y.: Numerical Verification Methods and Computer-Assisted Proofs for Partial Differential Equations. Springer Series in Computational Mathematics, Springer (2019). https://doi.org/10.1007/978-981-13-7669-6
48. Neumann, E., Ouaknine, J., Worrell, J.: On ranking function synthesis and termination for polynomial programs. In: Konnov, I., Kovács, L. (eds.) 31st International Conference on Concurrency Theory, CONCUR 2020, September 1–4, 2020, Vienna, Austria (Virtual Conference). LIPIcs, vol. 171, pp. 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.CONCUR.2020.15
49. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
50. Park, S., Ziegler, M.: Reliable degenerate matrix diagonalization. Tech. Rep. CS-TR-2018-415, KAIST (2018)
51. Pauly, A., Seon, D., Ziegler, M.: Computing Haar measures. In: Fernández, M., Muscholl, A. (eds.) 28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13–16, 2020, Barcelona, Spain. LIPIcs, vol. 152, pp. 34:1–34:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.CSL.2020.34
52. Pauly, A., Ziegler, M.: Relative computability and uniform continuity of relations. J. Log. Anal. **5**(7), 1–39 (2013)
53. Pour-El, M.B., Richards, J.I.: The wave equation with computable initial data such that its unique solution is not computable. Advances in Math. **39**, 215–239 (1981)
54. Rettinger, R.: Bloch's constant is computable. J. Univ. Comput. Sci. **14**(6), 896–907 (2008)
55. Ryu, S., Park, J., Park, J.: Toward analysis and bug finding in javascript web applications in the wild. IEEE Softw. **36**(3), 74–82 (2019). https://doi.org/10.1109/MS.2018.110113408
56. Schröder, M.: Admissible representations in computable analysis. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 471–480. Springer, Heidelberg (2006). https://doi.org/10.1007/11780342_48
57. Schröder, M.: Admissibly Represented Spaces and Qcb-Spaces. In: Handbook of Computability and Complexity in Analysis. TAC, pp. 305–346. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-59234-9_9
58. Selivanova, S., Steinberg, F., Thies, H., Ziegler, M.: Exact real computation of solution operators for linear analytic systems of partial differential equations. In: Boulier, F., England, M., Sadykov, T.M., Vorozhtsov, E.V. (eds.) CASC 2021. LNCS, vol. 12865, pp. 370–390. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85165-1_21
59. Specker, E.: The fundamental theorem of algebra in recursive analysis. In: Dejon, B., Henrici, P. (eds.) Constructive Aspects of the Fundamental Theorem of Algebra, pp. 321–329. Wiley-Interscience, London (1969)
60. Steinberg, F.: Complexity theory for spaces of integrable functions. Logical Methods in Computer Science 13(3), Paper No. 21, 39 (2017). https://doi.org/10.23638/LMCS-13(3:21)2017
61. Sun, S.-M., Zhong, N., Ziegler, M.: Computability of the solutions to Navier-Stokes equations via effective approximation. In: Du, D.-Z., Wang, J. (eds.) Complexity and Approximation. LNCS, vol. 12000, pp. 80–112. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-41672-0_7
62. Triebel, H.: Theory of Function Spaces I, II, III. Birkhäuser (1983, 1992, 2006). https://doi.org/10.1007/978-3-0346-0416-1

63. Weihrauch, K.: Computable Analysis. Springer, Berlin (2000). https://doi.org/10.1007/978-3-642-56999-9
64. Weihrauch, K.: The computable multi-functions on multi-represented sets are closed under programming. J. Univ. Comput. Sci. **14**(6), 801–844 (2008)
65. Weihrauch, K., Tavana-Roshandel, N.: Representations of measurable sets in computable measure theory. Logical Methods Comput. Sci. **10**, 3:7,21 (2014). https://doi.org/10.2168/LMCS-10(3:7)2014
66. Weihrauch, K., Zhong, N.: Is wave propagation computable or can wave computers beat the Turing machine? Proc. Lond. Math. Soc. **85**(2), 312–332 (2002)
67. Yap, C., Sagraloff, M., Sharma, V.: Analytic root clustering: a complete algorithm using soft zero tests. In: Bonizzoni, P., Brattka, V., Löwe, B. (eds.) CiE 2013. LNCS, vol. 7921, pp. 434–444. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39053-1_51
68. Yap, C.K.: On guaranteed accuracy computation. In: Geometric Computation, pp. 322–373. World Scientific Publishing, Singapore (2004)
69. Yu, J., Yap, C., Du, Z., Pion, S., Brönnimann, H.: The design of Core 2: a library for exact numeric computation in geometry and algebra. In: Fukuda, K., Hoeven, J., Joswig, M., Takayama, N. (eds.) ICMS 2010. LNCS, vol. 6327, pp. 121–141. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15582-6_24
70. Ziegler, M.: Real computation with least discrete advice: a complexity theory of nonuniform computability with applications to effective linear algebra. Ann. Pure Appl. Logic **163**(8), 1108–1139 (2012). https://doi.org/10.1016/j.apal.2011.12.030
71. Ziegler, M., Brattka, V.: Computability in linear algebra. Theoret. Comput. Sci. **326**(1–3), 187–211 (2004)